



PURE

JavaScript

Second Edition

**R. Allen Wyke
Jason D. Gilliam
Charlton Ting
Sean Michaels**

SAMS

201 West 103rd Street, Indianapolis, Indiana 46290

Pure JavaScript, Second Edition

Copyright © 2002 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32141-6

Library of Congress Catalog Card Number: 00-111512

Printed in the United States of America

First Printing: August 2001

04 03 02 01 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author(s) and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

ACQUISITIONS EDITOR

*Shelley Johnston
Markanday*

DEVELOPMENT EDITOR

Jonathan Steever

MANAGING EDITOR

Charlotte Clapp

PROJECT EDITOR

Elizabeth Finney

COPY EDITOR

Rhonda Tinch-Mize

INDEXER

Sandra Henselmeier

PROOFREADERS

*Antonio Reitz
Rebecca Martin
Debra Sexton*

TECHNICAL EDITOR

Andrew Watt

TEAM COORDINATOR

Amy Patton

MEDIA DEVELOPER

Dan Scherf

INTERIOR DESIGNER

Karen Ruggles

COVER DESIGNER

Aren Howell

PAGE LAYOUT

Ayanna Lacey

Overview

Introduction 1

PART I A PROGRAMMER'S OVERVIEW OF JAVASCRIPT

- 1 What Is JavaScript to a Programmer? 5
- 2 Details of the Language 31

PART II PROGRAMMING IN JAVASCRIPT

- 3 Programming Techniques 85
- 4 Client-Side Scripting 107
- 5 JavaScript on the Server-Side 157
- 6 Windows Scripting 179

PART III JAVASCRIPT REFERENCE

- 7 Core Language 193
- 8 Client-Side 563
- 9 Server-Side 1175

INDEX CD:2025

CD-ROM

- 10 DOM Core **CD:1403**
- 11 DOM HTML **CD:1479**
- 12 DOM Events **CD:1731**
- 13 DOM CSS **CD:1767**
- 14 JScript RunTime **CD:1823**
- 15 Windows Script Host **CD:1927**

Contents

INTRODUCTION	1
PART I A PROGRAMMER'S OVERVIEW OF JAVASCRIPT	
1 WHAT IS JAVASCRIPT TO A PROGRAMMER?	5
Welcome to JavaScript	6
Object-Based Technology	7
Object-Oriented Programming	8
Object Orientation of JavaScript	10
Object Access	16
Modular Programming	17
Security	18
What Security Measures Are in Place?	18
Data Tainting	20
Signed Scripts	21
Advantages of JavaScript	23
Platform Independence	24
Client-Side and Server-Side Versatility	24
When to Use JavaScript	25
Web Page Enhancements	26
Interactive E-Mail	26
Web-Based Applications	27
Windows Scripting	27
JavaScript Resources	28
General Information	28
Reference	29
Newsgroups	30
Moving On	30
2 DETAILS OF THE LANGUAGE	31
Things to Know about JavaScript Syntax	31
The Semicolon	31
Using the <script> Tag	32
Comments	33
Data Types and Variables	33
Numbers	34
Strings	35
Other Data Types	36
What to Know About Variables	36
Arrays	39
Operators	45

Arithmetic	45
String	49
Assignment	50
Logical	54
Comparison	55
Conditional	58
Bitwise	59
Precedence	61
Loops and Conditionals	62
Conditionals	62
Loops	65
Functions	76
Syntax	77
Call By Value Versus Call By Reference	77
Returning Values	79
Functions as Objects	80
Moving On	81

PART II PROGRAMMING IN JAVASCRIPT

3	PROGRAMMING TECHNIQUES	85
	Pattern Matching	85
	Defining the RegExp Object	86
	Defining Patterns	86
	Testing for Pattern Matches	88
	Events and Event Handlers	89
	Events	90
	Event Handlers	92
	Timers	95
	Exceptions	97
	throw	97
	try...catch	97
	Runtime Errors	99
	finally	100
	Debugging	101
	Microsoft Script Debugger	102
	The alert() Method	103
	Moving On	106
4	CLIENT-SIDE SCRIPTING	107
	Supporting Browsers and Their Issues	108
	Netscape Navigator	110
	Internet Explorer	111
	Opera	114
	HotJava	115

Extended Browser Functionality	115
Built-In Functions for Control.....	116
LiveConnect.....	120
Form Processing	124
The Forms Array	125
Accessing and Writing Information	128
Form Example	131
Cookie Handling.....	136
Reading Cookies	138
Writing Cookies.....	139
Windows in Action	139
The Parent/Child Relationship	139
Creating Windows	141
Limitations.....	142
Window Example	142
Dynamic Documents.....	145
The Old Way	146
The New Way	148
Cross Browser DHTML	151
XML-Based User Interface Language (XUL)	154
What Is XUL?	154
File Structure and Syntax	155
XUL and JavaScript	155
XUL Example	155
Moving On	156
5 JAVASCRIPT ON THE SERVER-SIDE	157
How Does Server-Side Differ from	
Client-Side?	158
Server-Side Environments	158
Programming Differences	158
Netscape's Server-Side JavaScript	159
Server Support	160
Compiling Your Scripts	160
The Server Side JavaScript Application Manager.....	161
Microsoft's ASP Environment	167
Server Support	168
Using the JScript Language.....	168
Checking Your Environment	169
Uses of Server-Side JavaScript.....	169
Within Enterprise Server	170
Within IIS	176
Moving On	178

6	WINDOWS SCRIPTING	179
	Windows Script Host	179
	Architecture	180
	Object Model	181
	Benefits and Limitations	182
	Windows Script Components	182
	Objects	182
	Methods of Executing Scripts	184
	Remote Scripting	187
	How to Use Remote Scripting.....	187
	Remote Scripting Example	188
	Moving On	189

PART III JAVASCRIPT REFERENCE

7	CORE LANGUAGE	193
	@_alpha.....	193
	@_jscript	194
	@_jscript_build	194
	@_jscript_version	195
	@_mac	196
	@_mc680x0.....	197
	@_PowerPC	197
	@_win16	198
	@_win32	198
	@_x86	199
	@cc_on	200
	@if.....	200
	@set	201
	- (Subtraction)	202
	- (Unary Negation)	203
	-- (Decrement)	204
	! (Logical NOT).....	205
	!= (Not Equal)	206
	!== (Non-Identity).....	207
	% (Modulus)	208
	%= (Modulus Assignment)	209
	& (Bitwise AND).....	210
	&& (Logical AND)	211
	&= (Bitwise AND Assignment)	212
	* (Multiplication).....	213
	*= (Multiplication Assignment)	213
	, (Comma)	214
	/ (Division).....	215
	/* / (Multi-line Comment).....	216

// (Comment)	216
/= (Division Assignment)	217
?: (Conditional)	218
^ (Bitwise Exclusive OR)	219
^= (Bitwise Exclusive OR Assignment)	220
(Bitwise OR)	221
(Logical OR)	222
= (Bitwise OR Assignment)	223
~ (Bitwise NOT)	224
+ (Addition)	225
++ (Increment)	226
+= (Addition Assignment)	227
< (Less Than)	228
<< (Shift Left)	229
<<= (Shift Left Assignment)	230
<= (Less Than or Equal)	231
= (Assignment)	232
-= (Subtraction Assignment)	232
== (Equal)	233
=== (Identity)	234
> (Greater Than)	235
>= (Greater Than or Equal)	236
>> (Shift Right with Sign)	237
>>= (Shift Right with Sign Assignment)	238
>>> (Shift Right Zero Fill)	239
>>>= (Shift Right Zero Fill Assignment)	240
abstract	241
ActiveXObject	241
Arguments	243
Arguments.callee	244
Arguments.caller	246
Arguments.length	247
Array	249
Array.concat()	251
Array.constructor	253
Array.index	253
Array.input	254
Array.join()	254
Array.lastIndex	255
Array.length	256
Array.pop()	257
Array.prototype	258
Array.push()	259
Array.reverse()	260
Array.shift()	261

Array.slice()	262
Array.sort()	263
Array.splice()	265
Array.toSource()	267
Array.toString()	268
Array.unshift()	269
Array.unwatch()	270
Array.valueOf()	271
Array.watch()	271
Automation	272
boolean	272
Boolean	273
Boolean.constructor	275
Boolean.prototype	275
Boolean.toSource()	276
Boolean.toString()	277
Boolean.unwatch()	278
Boolean.valueOf()	279
Boolean.watch()	279
break	280
byte	281
case	282
char	282
class	282
const	283
continue	283
Date	285
Date.constructor	289
Date.getDate()	289
Date.getDay()	290
Date.getFullYear()	291
Date.getHours()	292
Date.getMilliseconds()	292
Date.getMinutes()	293
Date.getMonth()	294
Date.getSeconds()	295
Date.getTime()	296
Date.getTimezoneOffset()	297
Date.getUTCDate()	298
Date.getUTCDay()	298
Date.getUTCFullYear()	300
Date.getUTCHours()	300
Date.getUTCMilliseconds()	301
Date.getUTCMinutes()	301
Date.getUTCMonth()	302

Date.getUTCSeconds()	304
Date.getVarDate()	304
Date.getYear()	305
Date.parse()	306
Date.prototype	307
Date.setDate()	308
Date.setFullYear()	309
Date.setHours()	310
Date.setMilliseconds()	310
Date.setMinutes()	311
Date.setMonth()	312
Date.setSeconds()	313
Date.setTime()	314
Date.setUTCDate()	314
Date.setUTCFullYear()	315
Date.setUTCHours()	316
Date.setUTCMilliseconds()	317
Date.setUTCMinutes()	318
Date.setUTCMonth()	318
Date.setUTCSeconds()	319
Date.setYear()	320
Date.toGMTString()	321
Date.toLocaleString()	322
Date.toSource()	322
Date.toString()	323
Date.toUTCString()	324
Date.unwatch()	325
Date.UTC()	326
Date.valueOf()	327
Date.watch	328
debugger	328
decodeURI()	329
decodeURIComponent()	330
default	331
delete	331
do	332
do...while	332
double	333
else	334
encodeURI()	334
encodeURIComponent()	335
enum	336
Enumerator	336
Enumerator.atEnd()	337
Enumerator.item()	338

Enumerator.moveFirst()	339
Enumerator.moveNext()	340
Error	341
Error.description	342
Error.message	343
Error.name	343
Error.number	344
escape()	345
eval()	346
export	347
extends	348
false	349
final	349
float	349
for	350
for...in	351
function	352
Function()	352
Function.apply()	353
Function.arguments	354
Function.arity	355
Function.call()	356
Function.callee	357
Function.caller	358
Function.constructor	359
Function.length	360
Function.prototype	360
Function.toSource()	361
Function.toString()	362
Function.valueOf()	363
GetObject	364
Global	365
Global.decodeURI()	366
Global.decodeURIComponent()	367
Global.encodeURI()	368
Global.encodeURIComponent()	369
Global.escape()	370
Global.eval()	371
Global.Infinity	373
Global.isFinite()	374
Global.isNaN()	375
Global.NaN	376
Global.Number()	376
Global.parseFloat()	377
Global.parseInt()	378

Global.String()	379
Global.unescape()	380
Global.undefined	380
goto	381
if	381
if...else	382
implements	383
import	383
in	384
Infinity	385
instanceof	386
int	387
interface	387
isFinite()	387
isNaN()	388
java	389
javaArray	390
javaArray.length	391
javaArray.toString	392
javaClass	392
javaObject	393
javaPackage	394
JSException	395
JSException.getWrappedException()	396
JSObject	396
JSObject.call()	398
JSObject.equals()	399
JSObject.eval()	400
JSObject.getMember()	401
JSObject.getSlot()	403
JSObject.getWindow()	403
JSObject.removeMember()	404
JSObject.setMember()	405
JSObject.setSlot()	405
JSObject.toString()	406
label	406
long	407
Math()	408
Math.abs()	409
Math.acos()	410
Math.asin()	411
Math.atan()	412
Math.atan2()	413
Math.ceil()	414
Math.cos()	415

Math.E	416
Math.exp()	417
Math.floor()	418
Math.LN10	419
Math.LN2	420
Math.log()	420
Math.LOG10E	421
Math.LOG2E	422
Math.max()	423
Math.min()	424
Math.PI	425
Math.pow()	426
Math.random()	427
Math.round()	428
Math.sin()	429
Math.sqrt()	430
Math.SQRT1_2	431
Math.SQRT2	432
Math.tan()	432
Math.toSource()	433
Math.toString()	434
Math.unwatch()	435
Math.watch()	435
NaN	436
native	436
netscape	437
new	437
null	438
Number()	438
Number()	439
Number.constructor	440
Number.MAX_VALUE	441
Number.MIN_VALUE	442
Number.NaN	443
Number.NEGATIVE_INFINITY	443
Number.POSITIVE_INFINITY	444
Number.prototype	445
Number.toExponential()	446
Number.toFixed()	447
Number.toLocaleString()	447
Number.toPrecision()	448
Number.toSource()	448
Number.toString()	449
Number.unwatch()	449
Number.valueOf()	450

Number.watch()	451
Object()	452
Object.constructor	453
Object.eval()	454
Object.prototype	455
Object.toSource()	455
Object.toString()	456
Object.unwatch()	457
Object.valueOf()	458
Object.watch()	458
package	459
Packages	460
Packages.className	461
Packages.java	461
Packages.netscape	462
Packages.sun	463
parseFloat()	464
parseInt()	465
private	465
protected	466
public	466
RegExp()	466
RegExp,\$*	469
RegExp,\$&	470
RegExp,\$_	470
RegExp,\$`	471
RegExp,\$'	472
RegExp,\$+	473
RegExp,\$1,\$2,..,\$9	474
RegExp.constructor()	475
RegExp.compile()	476
RegExp.exec()	477
RegExp.global	478
RegExp.ignoreCase	479
RegExp.input	480
RegExp.lastIndex	481
RegExp.lastMatch	481
RegExp.lastParen	482
RegExp.leftContext	483
RegExp.multiline	484
RegExp.prototype	485
RegExp.rightContext	486
RegExp.source	487
RegExp.test()	487
RegExp.toSource()	488

RegExp.toString()	489
RegExp.unwatch()	490
RegExp.valueOf()	491
RegExp.watch()	492
return	493
ScriptEngine	494
ScriptEngineBuildVersion	494
ScriptEngineMajorVersion	495
ScriptEngineMinorVersion	495
short	496
static	496
String (Function)	497
String (Object)	497
String.anchor()	501
String.big()	502
String.blink()	503
String.bold()	503
String.charAt()	504
String.charCodeAtAt()	505
String.concat()	507
String.constructor	507
String.fixed()	508
String.fontcolor()	509
String.fontSize()	509
String.fromCharCode()	510
String.indexOf()	511
String.italics()	512
String.lastIndexOf()	513
String.length	514
String.link()	515
String.localeCompare()	515
String.match()	516
String.prototype	517
String.replace()	519
String.search()	520
String.slice()	521
String.small()	522
String.split()	522
String.strike()	524
String.sub()	525
String.substr()	526
String.substring()	527
String.sup()	528
String.toLocaleLowerCase()	528
String.toLocaleUpperCase()	529

String.toLowerCase()	530
String.toSource()	531
String.toString()	532
String.toUpperCase()	533
String.unwatch()	533
String.valueOf()	534
String.watch()	535
sun	536
super	537
switch	537
synchronized	539
this	539
throw	541
throws	543
transient	543
true	544
try...catch...finally	544
typeof	546
undefined	548
unescape()	548
var	549
VBAArray	551
VBAArray.dimensions()	552
VBAArray.getItem()	553
VBAArray.lbound()	555
VBAArray.toArray()	556
VBAArray.ubound()	557
void	559
volatile	559
while	560
with	561
8 CLIENT-SIDE	563
Anchor	564
Anchor.name	566
Anchor.text	567
Anchor.unwatch()	568
Anchor.watch()	569
Anchor.x	570
Anchor.y	571
Applet	572
Area	573
Area.handleEvent()	576
Area.hash	578
Area.host	580

Area.hostname	582
Area.href	583
Area.onDbClick	584
Area.onMouseOut	585
Area.onMouseOver	586
Area.pathname	587
Area.port	588
Area.protocol	589
Area.search	590
Area.target	591
Area.unwatch()	594
Area.watch()	596
Button	597
Button.blur()	599
Button.click()	600
Button.focus()	601
Button.form	602
Button.handleEvent()	603
Button.name	605
Button.onBlur	606
Button.onClick	606
Button.onFocus	607
Button.onMouseDown	608
Button.onMouseUp	608
Button.type	609
Button.unwatch()	610
Button.value	612
Button.watch()	613
Checkbox	614
Checkbox.blur()	617
Checkbox.checked	618
Checkbox.click()	620
Checkbox.defaultChecked	621
Checkbox.focus()	622
Checkbox.form	623
Checkbox.handleEvent()	624
Checkbox.name	626
Checkbox.onBlur	627
Checkbox.onClick	628
Checkbox.onFocus	630
Checkbox.type	631
Checkbox.unwatch()	632
Checkbox.value	634
Checkbox.watch()	635
Document	636

document.alinkColor	637
document.all	638
document.all.item()	639
document.all.tags()	640
document.anchors	641
document.anchors.length	642
document.applets	643
document.applets.length	644
document.bgColor	645
document.captureEvents()	646
document.classes	648
document.close()	651
document.contextual()	652
document.cookie	652
document.domain	653
document.embeds	654
document.embeds.length	655
document.fgColor	655
document.formName	656
document.forms	657
document.forms.length	658
document.getSelection()	659
document.handleEvent()	660
document.ids	661
document.images	664
document.images.length	665
document.lastModified	666
document.layers	667
document.layers.length	668
document.linkColor	669
document.links	670
document.links.length	671
document.onClick	672
document.onDbClick	673
document.onKeyDown	674
document.onKeyPress	675
document.onKeyUp	676
document.onLoad	677
document.onMouseDown	677
document.onMouseUp	678
document.onUnLoad	679
document.open()	680
document.plugins	681
document.plugins.length	682
document.referrer	682

document.releaseEvents()	683
document.routeEvent()	685
document.tags	687
document.title	690
document.unwatch()	690
document.URL	692
document.vlinkColor	692
document.watch()	693
document.write()	694
document.writeln()	695
Embed	697
Event	698
event.data	701
event.height	702
event.layerX	703
event.layerY	704
event.modifiers	705
event.pageX	707
event.pageY	708
event.screenX	709
event.screenY	711
event.target	712
event.type	713
event.unwatch()	715
event.watch()	716
event.which	718
event.width	718
Event.ABORT	720
Event.BLUR	721
Event.CHANGE	722
Event.CLICK	723
Event.DBLCLICK	724
Event.DRAGDROP	725
Event.ERROR	726
Event.FOCUS	728
Event.KEYDOWN	729
Event.KEYPRESS	730
Event.KEYUP	731
Event.LOAD	732
Event.MOUSEDOWN	733
Event.MOUSEMOVE	734
Event.MOUSEOUT	735
Event.MOUSEOVER	736
Event.MOUSEUP	737
Event.MOVE	738

Event.RESET	739
Event.RESIZE	741
Event.SELECT	742
Event.SUBMIT	743
Event.UNLOAD	744
FileUpload	745
FileUpload.blur()	747
FileUpload.focus()	748
FileUpload.form	749
FileUpload.handleEvent()	750
FileUpload.name	752
FileUpload.onBlur	753
FileUpload.onChange	754
FileUpload.onFocus	755
FileUpload.select()	756
FileUpload.type	757
FileUpload.unwatch()	758
FileUpload.value	759
FileUpload.watch()	760
Form	761
Form.action	763
Form.elements	764
Form.elements.length	765
Form.encoding	766
Form.handleEvent()	767
Form.length	768
Form.method	769
Form.name	770
Form.onReset	771
Form.onSubmit	772
Form.reset()	773
Form.submit()	774
Form.target	775
Form.unwatch()	776
Form.watch()	777
Frame	778
Frame.blur()	779
Frame.clearInterval()	780
Frame.clearTimeout()	781
Frame.document	781
Frame.focus()	782
Frame.frames	782
Frame.length	783
Frame.name	784
Frame.onBlur	784

Frame.onFocus	785
Frame.onMove.....	786
Frame.onResize	786
Frame.parent	787
Frame.print()	787
Frame.self	788
Frame.setInterval().....	788
Frame.setTimeout()	789
Frame.top.....	790
Frame.unwatch()	790
Frame.watch()	791
Frame.window	792
Hidden	793
Hidden.form.....	794
Hidden.name	795
Hidden.type	795
Hidden.unwatch()	796
Hidden.value	797
Hidden.watch()	798
History	799
History.back()	800
History.current	800
History.forward()	801
History.go()	802
History.length	802
History.next	803
History.previous.....	804
History.unwatch()	805
History.watch()	806
Image	807
Image.border	809
Image.complete	810
Image.handleEvent()	811
Image.height	811
Image.hspace	812
Image.lowsrc	813
Image.name	814
Image.onAbort.....	814
Image.onError	815
Image.onKeyDown	815
Image.onKeyPress	816
Image.onKeyUp.....	816
Image.onLoad	816
Image.src	817
Image.unwatch()	818

Image.vspace	819
Image.watch()	819
Image.width	820
Layer	821
Layer.above	823
Layer.background	824
Layer.below	825
Layer.bgColor	826
Layer.captureEvents()	827
Layer.clip.bottom	828
Layer.clip.height	829
Layer.clip.left	830
Layer.clip.right	831
Layer.clip.top	832
Layer.clip.width	833
Layer.document	834
Layer.handleEvent()	835
Layer.left	837
Layer.load()	838
Layer.moveAbove()	838
Layer.moveBelow()	839
Layer.moveBy()	840
Layer.moveTo()	841
Layer.moveToAbsolute()	842
Layer.name	843
Layer.onBlur	844
Layer.onFocus	845
Layer.onLoad	846
Layer.onMouseOut	847
Layer.onMouseOver	848
Layer.pageX	849
Layer.pageY	850
Layer.parentLayer	851
Layer.releaseEvents()	852
Layer.resizeBy()	853
Layer.resizeTo()	854
Layer.routeEvent()	855
Layer.siblingAbove	856
Layer.siblingBelow	857
Layer.src	858
Layer.top	859
Layer.unwatch()	860
Layer.visibility	861
Layer.watch()	862
Layer.zIndex	864

Link	864
Link.handleEvent()	866
Link.hash	867
Link.host	868
Link.hostname	869
Link.href	870
Link.onClick	871
Link.onDbClick	871
Link.onKeyDown	872
Link.onKeyPress	873
Link.onKeyUp	874
Link.onMouseDown	874
Link.onMouseOut	875
Link.onMouseOver	876
Link.onMouseUp	877
Link.pathname	877
Link.port	878
Link.protocol	879
Link.search	880
Link.target	881
Link.text	882
Link.unwatch()	883
Link.watch()	884
Location	885
Location.hash	886
Location.host	887
Location.hostname	889
Location.href	890
Location.pathname	891
Location.port	892
Location.protocol	893
Location.reload()	894
Location.replace()	895
Location.search	895
Location.unwatch()	896
Location.watch()	898
MimeType	899
MimeType.description	900
MimeType.enabledPlugin	901
MimeType.suffixes	901
MimeType.type	902
MimeType.unwatch()	903
MimeType.watch()	903
navigator	904
navigator.appCodeName	905

navigator.appName	906
navigator.appVersion	906
navigator.javaEnabled()	907
navigator.language	908
navigator.mimeTypes	908
navigator.platform	909
navigator.plugins	910
navigator.plugins.refresh()	911
navigator.preference()	911
navigator.taintEnabled()	912
navigator.userAgent	913
navigator.unwatch()	914
navigator.watch()	914
Option	914
Option.defaultSelected	916
Option.selected	917
Option.text	918
Option.unwatch()	919
Option.value	920
Option.watch()	921
Password	922
Password.blur()	923
Password.defaultValue	924
Password.focus()	925
Password.form	925
Password.handleEvent()	926
Password.name	927
Password.onBlur	928
Password.onFocus	929
Password.select()	930
Password.type	930
Password.unwatch()	931
Password.value	932
Password.watch()	933
Plugin	934
Plugin.description	935
Plugin.filename	935
Plugin.length	936
Plugin.name	937
Plugin.unwatch()	937
Plugin.watch()	938
Radio	938
Radio.blur()	939
Radio.checked	940
Radio.click()	941

Radio.defaultChecked	942
Radio.focus()	943
Radio.form	944
Radio.handleEvent()	944
Radio.name	946
Radio.onBlur	946
Radio.onClick	947
Radio.onFocus()	948
Radio.type	949
Radio.unwatch()	949
Radio.value	950
Radio.watch()	951
Reset	952
Reset.blur()	953
Reset.click()	954
Reset.focus()	955
Reset.form	956
Reset.handleEvent()	957
Reset.name	958
Reset.onBlur	959
Reset.onClick	960
Reset.onFocus	961
Reset.type	962
Reset.unwatch()	962
Reset.value	964
Reset.watch()	965
screen	967
screen.availHeight	968
screen.availLeft	969
screen.availTop	969
screen.availWidth	970
screen.colorDepth	971
screen.height	971
Screen.pixelDepth	972
screen.unwatch()	972
screen.watch()	973
screen.width	974
Select	974
Select.blur()	976
Select.focus()	978
Select.form	979
Select.handleEvent()	980
Select.length	982
Select.name	983
Select.onBlur	984

Select.onChange	985
Select.onFocus	986
Select.options.....	987
Select.options.length	989
Select.options.selectedIndex	989
Select.options.value	990
Select.selectedIndex	991
Select.type	991
Select.unwatch()	992
Select.watch()	993
Style	994
Style.align	997
Style.backgroundColor	998
Style.backgroundImage	999
Style.borderBottomWidth	1000
Style.borderColor	1001
Style.borderLeftWidth	1001
Style.borderRightWidth.....	1002
Style.borderStyle	1003
Style.borderTopWidth	1004
Style.borderWidths()	1004
Style.clear	1006
Style.color	1007
Style.display	1007
Style.fontFamily	1008
Style.fontSize.....	1009
Style.fontStyle	1010
Style.fontWeight	1011
Style.lineHeight	1012
Style.listStyleType.....	1013
Style.marginBottom.....	1014
Style.marginLeft	1015
Style.marginRight	1016
Style.margins()	1016
Style.marginTop.....	1018
Style.paddingBottom	1018
Style.paddingLeft	1019
Style.paddingRight	1020
Style.paddings().....	1021
Style.paddingTop.....	1022
Style.textAlign	1023
Style.textDecoration	1024
Style.textIndent	1024
Style.textTransform	1025
Style.unwatch()	1026

Style.watch()	1027
Style.whiteSpace	1028
Style.width	1028
Submit	1029
Submit.blur()	1031
Submit.click()	1032
Submit.focus()	1034
Submit.form	1034
Submit.handleEvent()	1036
Submit.name	1037
Submit.onBlur	1038
Submit.onClick	1039
Submit.onFocus	1040
Submit.type	1041
Submit.unwatch()	1042
Submit.value	1043
Submit.watch()	1044
taint()	1045
Text	1046
Text.blur()	1048
Text.defaultValue	1049
Text.focus()	1050
Text.form	1051
Text.handleEvent()	1052
Text.name	1053
Text.onBlur	1054
Text.onChange	1055
Text.onFocus	1056
Text.onSelect	1057
Text.select()	1058
Text.type	1059
Text.unwatch()	1060
Text.value	1061
Text.watch()	1062
Textarea	1063
Textarea.blur()	1065
Textarea.defaultValue	1066
Textarea.focus()	1067
Textarea.form	1068
Textarea.handleEvent()	1070
Textarea.name	1071
Textarea.onBlur	1072
Textarea.onChange	1073
Textarea.onFocus	1074
Textarea.onKeyDown	1075

Textarea.onKeyPress	1076
Textarea.onKeyUp	1077
Textarea.onSelect	1078
Textarea.select()	1079
Textarea.type	1080
Textarea.unwatch()	1081
Textarea.value	1082
Textarea.watch()	1083
untaint()	1084
Window	1085
window.alert()	1090
window.atob()	1091
window.back()	1092
window.blur()	1092
window.btoa()	1094
window.captureEvents()	1094
window.clearInterval()	1096
window.clearTimeout()	1098
window.close()	1099
window.closed	1100
window.confirm()	1102
window.crypto	1103
window.crypto.random()	1104
window.crypto.signText()	1104
window.defaultStatus	1105
window.disableExternalCapture()	1106
window.document	1107
window.enableExternalCapture()	1110
window.find()	1111
window.focus()	1112
window.forward()	1113
window.frames	1114
window.frames.length	1115
window.handleEvent()	1115
window.history	1116
window.home()	1118
window.innerHeight	1118
window.innerWidth	1120
window.length	1121
window.location	1122
window.locationbar	1122
window.locationbar.visible	1123
window.menuubar	1123
window.menuubar.visible	1124
window.moveBy()	1125

window.moveTo()	1126
window.name	1127
window.offscreenBuffering	1129
window.onBlur	1129
window.onDragDrop	1130
window.onError	1130
window.onFocus	1131
window.onLoad	1132
window.onMove	1132
window.onResize	1133
window.onUnload	1133
window.open()	1134
window.opener	1137
window.outerHeight	1139
window.outerWidth	1140
window.pageXOffset	1142
window.pageYOffset	1143
window.parent	1144
window.personalbar	1144
window.personalbar.visible	1145
window.print()	1146
window.prompt()	1147
window.releaseEvents()	1148
window.resizeBy()	1150
window.resizeTo()	1151
window.routeEvent()	1152
window.screenX	1154
window.screenY	1155
window.scroll()	1156
window.scrollbars	1157
window.scrollbars.visible	1158
window.scrollBy()	1158
window.scrollTo()	1160
window.self	1161
window.setHotKeys()	1162
window.setInterval()	1162
window.setResizable()	1164
window.setTimeout()	1164
window.setZOptions()	1165
window.status	1166
window.statusbar	1167
window.statusbar.visible	1167
window.stop()	1168
window.toolbar	1169
window.toolbar.visible	1169

window.top.....	1170
window.unwatch()	1171
window.watch()	1172
window.window	1173
9 SERVER-SIDE	1175
addClient()	1176
addResponseHeader()	1176
Application	1177
Application.Application_OnEnd	1178
Application.Application_OnStart	1179
Application.Contents	1179
Application.Contents.Remove()	1180
Application.Contents.RemoveAll()	1180
Application.Lock()	1181
Application.StaticObjects	1182
Application.Unlock()	1182
ASPErrors	1183
ASPErrors.ASPCode	1184
ASPErrors.ASPDescription	1184
ASPErrors.Category	1185
ASPErrors.Column	1185
ASPErrors.Description	1185
ASPErrors.File	1186
ASPErrors.Line	1186
ASPErrors.Number	1187
ASPErrors.Source	1187
blob() (Function)	1188
blob (Object)	1189
blob.blobImage()	1190
blob.blobLink()	1191
callC()	1192
client	1193
client.destroy()	1195
client.expiration()	1196
client.property	1196
client.unwatch()	1197
client.watch()	1198
Connection	1199
Connection.beginTransaction()	1201
Connection.commitTransaction()	1202
Connection.connected()	1203
Connection.cursor()	1204
Connection.execute()	1205
Connection.majorErrorCode()	1205

Connection.majorErrorMessage()	1206
Connection.minorErrorCode()	1207
Connection.minorErrorMessage()	1208
Connection.prototype	1209
Connection.release()	1210
Connection.rollbackTransaction()	1211
Connection.SQLTable()	1212
Connection.storedProc()	1213
Connection.toString()	1215
Connection.unwatch()	1216
Connection.watch()	1218
Cursor	1219
Cursor.close()	1221
Cursor.columnName	1222
Cursor.columnName()	1222
Cursor.columns()	1224
Cursor.deleteRow()	1225
Cursor.insertRow()	1226
Cursor.next()	1227
Cursor.prototype	1228
Cursor.unwatch()	1229
Cursor.updateRow()	1231
Cursor.watch()	1232
database	1233
database.beginTransaction()	1235
database.commitTransaction()	1236
database.connect()	1237
database.connected()	1239
database.cursor()	1240
database.disconnect()	1241
database.execute()	1242
database.majorErrorCode()	1242
database.majorErrorMessage()	1243
database.minorErrorCode()	1244
database.minorErrorMessage()	1245
database.prototype	1246
database.rollbackTransaction()	1247
database.SQLTable()	1248
database.storedProc()	1249
database.storedProcArgs()	1250
database.toString()	1251
database.unwatch()	1253
database.watch()	1254
DbPool	1255
DbPool.connect()	1258

DbPool.connected()	1260
DbPool.connection()	1261
DbPool.DbPool()	1262
DbPool.disconnect()	1263
DbPool.majorErrorCode()	1264
DbPool.majorErrorMessage()	1264
DbPool.minorErrorCode()	1265
DbPool.minorErrorMessage()	1266
DbPool.prototype	1267
DbPool.storedProcArgs()	1268
DbPool.toString()	1269
DBPool.unwatch()	1270
DBPool.watch()	1272
debug()	1273
deleteResponseHeader()	1274
File()	1274
File.byteToString()	1277
File.clearError()	1278
File.close()	1279
File.constructor	1279
File.eof()	1280
File.error()	1281
File.exists()	1282
File.flush()	1283
File.getLength()	1284
File.getPosition()	1284
File.open()	1285
File.prototype	1286
File.read()	1288
File.readByte()	1288
File.readLine()	1289
File.setPosition()	1290
File.stringToByte()	1291
File.unwatch()	1292
File.watch()	1294
File.write()	1295
File.writeByte()	1296
File.writeln()	1297
flush()	1298
getOptionValue()	1299
getOptionValueCount()	1300
Lock()	1300
Lock.constructor	1301
Lock.isValid()	1302
Lock.lock()	1303

Lock.prototype.....	1303
Lock.unlock()	1305
Lock.unwatch()	1305
Lock.watch()	1307
ObjectContext	1308
ObjectContext.OnTransactionAbort	1309
ObjectContext.OnTransactionCommit	1310
ObjectContext.SetAbort()	1310
ObjectContext.SetComplete()	1311
project	1312
project.lock()	1313
project.unlock()	1313
project.unwatch()	1314
project.watch()	1315
redirect()	1316
registerCFunction()	1317
Request	1318
Request.BinaryRead()	1318
Request.ClientCertificate.....	1319
Request.Cookies	1321
Request.Form.....	1321
Request.Form.Form.Count	1322
Request.QueryString	1323
Request.QueryString.Count.....	1323
Request.ServerVariables	1324
Request.TotalBytes	1326
request	1327
request.agent	1328
request.formKey	1328
request.imageX	1329
request.imageY	1330
request.ip	1331
request.method.....	1331
request.protocol	1332
request.unwatch()	1333
request.watch()	1334
Response	1334
Response.AddHeader()	1335
Response.AppendToLog()	1336
Response.BinaryWrite()	1336
Response.Buffer	1337
Response.CacheControl.....	1337
Response.Charset.....	1338
Response.Clear()	1338
Response.ContentType	1339

Response.Cookies	1339
Response.End()	1340
Response.Expires.....	1340
Response.ExpiresAbsolute	1341
Response.Flush()	1341
Response.IsClientConnected	1342
Response.PICS	1342
Response.Redirect().....	1343
Response.Status	1343
Response.Write()	1344
Resultset.....	1344
Resultset.close().....	1346
Resultset.columnName()	1347
Resultset.columns()	1348
Resultset.next()	1349
Resultset.prototype	1350
Resultset.unwatch()	1351
Resultset.watch()	1353
ScriptingContext	1354
SendMail	1354
SendMail.Bcc	1357
SendMail.Body	1357
SendMail.Cc	1358
SendMail.constructor.....	1358
SendMail.errorCode()	1359
SendMail.errorMessage()	1360
SendMail.Errorsto	1361
SendMail.From	1361
SendMail.Organization	1362
SendMail.prototype	1362
SendMail.Replyto	1363
SendMail.send().....	1364
SendMail.Smtpserver	1365
SendMail.Subject.....	1365
SendMail.To	1366
SendMail.unwatch()	1366
SendMail.watch()	1367
Server.....	1368
Server.CreateObject()	1369
Server.Execute().....	1369
Server.GetLastError()	1370
Server.HTMLEncode()	1370
Server.MapPath()	1371
Server.ScriptTimeout	1371
Server.Transfer()	1372

Server.URLEncode()	1372
server	1373
server.host	1374
server.hostname	1374
server.lock()	1375
server.port	1375
server.protocol	1376
server.unlock()	1376
server.unwatch().....	1377
server.watch().....	1378
Session	1378
Session.Abandon().....	1379
Session.CodePage	1380
Session.Contents	1381
Session.Contents.Remove().....	1381
Session.Contents.RemoveAll()	1382
Session.LCID.....	1383
Session.SessionID	1383
Session.Session_OnEnd	1384
Session.Session_OnStart	1384
Session.StaticObjects.....	1385
Session.Timeout.....	1385
ssjs_generateClientID()	1385
ssjs_getCGIVariable()	1386
ssjs_getClientID()	1387
Stproc.....	1388
Stproc.close().....	1389
Stproc.outParamCount()	1390
Stproc.outParameters()	1391
Stproc.prototype	1392
Stproc.resultSet()	1393
Stproc.returnValue()	1395
Stproc.unwatch()	1395
Stproc.watch()	1397
write()	1398

CD-ROM ONLY CHAPTERS

10 DOM CORE

(CD-ROM) 1403

Attr	CD:1403
Attr.Name	CD:1404
Attr.specified	CD:1404
Attr.value.....	CD:1405
Attr.ownerElement	CD:1406

CDATASection	CD:1406
CharacterData	CD:1407
CharacterData.data	CD:1408
CharacterData.length	CD:1408
CharacterData.substringData()	CD:1409
CharacterData.appendData()	CD:1409
CharacterData.insertData()	CD:1410
CharacterData.deleteData()	CD:1411
CharacterData.replaceData()	CD:1411
Comment	CD:1412
Document	CD:1412
Document.createAttribute()	CD:1414
Document.createAttributeNS()	CD:1414
Document.createCDATASection()	CD:1415
Document.createComment()	CD:1416
Document.createDocumentFragment()	CD:1416
Document.createElement()	CD:1417
Document.createElementNS()	CD:1417
Document.createEntityReference()	CD:1418
Document.createProcessingInstruction()	CD:1419
Document.createTextNode()	CD:1419
Document.doctype	CD:1420
Document.documentElement	CD:1421
Document.getElementById()	CD:1421
Document.getElementsByTagName()	CD:1422
Document.getElementsByTagNameNS()	CD:1422
Document.implementation	CD:1423
Document.importNode()	CD:1424
DocumentFragment	CD:1424
DocumentType	CD:1425
DocumentType.entities	CD:1426
DocumentType.internalSubset	CD:1426
DocumentType.name	CD:1427
DocumentType.notations	CD:1427
DocumentType.publicId	CD:1428
DocumentType.systemId	CD:1429
DOMException	CD:1429
DOMException.code	CD:1430
DOMImplementation	CD:1431
DOMImplementation.createDocument()	CD:1432
DOMImplementation.createDocumentType()	CD:1433
DOMImplementation.hasFeature()	CD:1433
Element	CD:1434
Element.getAttribute()	CD:1435
Element.getAttributeNode()	CD:1436

Element.getAttributeNodeNS()	CD:1436
Element.getAttributeNS()	CD:1437
Element.getElementsByTagName()	CD:1438
Element.getElementsByTagNameNS()	CD:1438
Element.hasAttribute()	CD:1439
Element.hasAttributeNS()	CD:1439
Element.removeAttribute()	CD:1440
Element.removeAttributeNode()	CD:1440
Element.removeAttributeNS()	CD:1441
Element.setAttribute()	CD:1442
Element.setAttributeNode()	CD:1442
Element.setAttributeNodeNS()	CD:1443
Element.setAttributeNS()	CD:1444
Element.tagName	CD:1444
Entity	CD:1445
Entity.notationName	CD:1445
Entity.publicId	CD:1446
Entity.systemId	CD:1446
EntityReference	CD:1447
NamedNodeMap	CD:1447
NamedNodeMap.getNamedItem()	CD:1448
NamedNodeMap.getNamedItemNS()	CD:1449
NamedNodeMap.item()	CD:1449
NamedNodeMap.length	CD:1450
NamedNodeMap.removeNamedItem()	CD:1451
NamedNodeMap.removeNamedItemNS()	CD:1451
NamedNodeMap.setNamedItem()	CD:1452
NamedNodeMap.setNamedItemNS()	CD:1453
Node	CD:1453
Node.attributes	CD:1456
Node.appendChild()	CD:1457
Node.childNodes	CD:1457
Node.cloneNode()	CD:1458
Node.firstChild	CD:1458
Node.hasAttributes()	CD:1459
Node.hasChildNodes()	CD:1460
Node.insertBefore()	CD:1460
Node.isSupported()	CD:1461
Node.lastChild	CD:1462
Node.localName	CD:1462
Node.namespaceURI	CD:1463
Node.nextSibling	CD:1463
Node.nodeName	CD:1464
Node.nodeType	CD:1465
Node.nodeValue	CD:1465

Node.normalize()	CD:1466
Node.ownerDocument	CD:1466
Node.parentNode	CD:1467
Node.prefix	CD:1468
Node.previousSibling	CD:1468
Node.removeChild()	CD:1469
Node.replaceChild()	CD:1469
Notation	CD:1470
Notation.publicId	CD:1471
Notation.systemId	CD:1471
ProcessingInstruction	CD:1472
ProcessingInstruction.data	CD:1472
ProcessingInstruction.target	CD:1473
Text	CD:1474
Text.splitText()	CD:1474

11 DOM HTML (CD-ROM) 1479

HTMLAnchorElement	CD:1480
HTMLAnchorElement.accessKey	CD:1481
HTMLAnchorElement.blur()	CD:1482
HTMLAnchorElement.charset	CD:1482
HTMLAnchorElement.coords	CD:1483
HTMLAnchorElement.focus()	CD:1484
HTMLAnchorElement.href	CD:1484
HTMLAnchorElement.hrefLang	CD:1485
HTMLAnchorElement.name	CD:1485
HTMLAnchorElement.rel	CD:1486
HTMLAnchorElement.rev	CD:1486
HTMLAnchorElement.shape	CD:1487
HTMLAnchorElement.tabIndex	CD:1488
HTMLAnchorElement.target	CD:1488
HTMLAnchorElement.type	CD:1489
HTMLAppletElement	CD:1489
HTMLAppletElement.align	CD:1490
HTMLAppletElement.alt	CD:1491
HTMLAppletElement.archive	CD:1492
HTMLAppletElement.code	CD:1492
HTMLAppletElement.codeBase	CD:1493
HTMLAppletElement.height	CD:1493
HTMLAppletElement.hspace	CD:1494
HTMLAppletElement.name	CD:1495
HTMLAppletElement.object	CD:1495
HTMLAppletElement.vspace	CD:1496
HTMLAppletElement.width	CD:1496
HTMLAreaElement	CD:1497

HTMLAreaElement.accessKey	CD:1498
HTMLAreaElement.alt	CD:1498
HTMLAreaElement.coords	CD:1499
HTMLAreaElement.href	CD:1500
HTMLAreaElement.noHref	CD:1500
HTMLAreaElement.shape	CD:1501
HTMLAreaElement.tabIndex	CD:1501
HTMLAreaElement.target	CD:1502
HTMLBaseElement	CD:1503
HTMLBaseElement.href	CD:1503
HTMLBaseElement.target	CD:1504
HTMLBaseFontElement	CD:1505
HTMLBaseFontElement.color	CD:1505
HTMLBaseFontElement.face	CD:1506
HTMLBaseFontElement.size	CD:1507
HTMLBodyElement	CD:1507
HTMLBodyElement.aLink	CD:1508
HTMLBodyElement.background	CD:1509
HTMLBodyElement.bgColor	CD:1509
HTMLBodyElement.link	CD:1510
HTMLBodyElement.text	CD:1511
HTMLBodyElement.vLink	CD:1511
HTMLBRElement	CD:1512
HTMLBRElement.clear	CD:1512
HTMLButtonElement	CD:1513
HTMLButtonElement.accessKey	CD:1514
HTMLButtonElement.disabled	CD:1514
HTMLButtonElement.form	CD:1515
HTMLButtonElement.name	CD:1516
HTMLButtonElement.tabIndex	CD:1516
HTMLButtonElement.type	CD:1517
HTMLButtonElement.value	CD:1517
HTMLCollection	CD:1518
HTMLCollection.item()	CD:1519
HTMLCollection.length	CD:1520
HTMLCollection.namedItem()	CD:1520
HTMLDirectoryElement	CD:1521
HTMLDirectoryElement.compact	CD:1522
HTMLDivElement	CD:1522
HTMLDivElement.align	CD:1523
HTMLDListElement	CD:1524
HTMLDListElement.compact	CD:1524
HTMLDocument	CD:1525
HTMLDocument.anchors	CD:1526
HTMLDocument.applets	CD:1527

HTMLDocument.body	CD:1528
HTMLDocument.close().....	CD:1528
HTMLDocument.cookie	CD:1529
HTMLDocument.domain	CD:1530
HTMLDocument.forms	CD:1530
HTMLDocument.getElementsByName()	CD:1531
HTMLDocument.images	CD:1532
HTMLDocument.links	CD:1533
HTMLDocument.open()	CD:1533
HTMLDocument.referrer	CD:1534
HTMLDocument.title	CD:1535
HTMLDocument.URL	CD:1535
HTMLDocument.write().....	CD:1536
HTMLDocument.writeln()	CD:1537
HTMLDOMImplementation.....	CD:1537
HTMLDOMImplementation.createHTMLDocument()	CD:1538
HTMLElement	CD:1539
HTMLElement.className	CD:1540
HTMLElement.dir	CD:1540
HTMLElement.id	CD:1541
HTMLElement.lang	CD:1541
HTMLElement.title.....	CD:1542
HTMLFieldSetElement.....	CD:1543
HTMLFieldSetElement.form	CD:1543
HTMLFontElement.....	CD:1544
HTMLFontElement.color	CD:1545
HTMLFontElement.face	CD:1545
HTMLFontElement.size	CD:1546
HTMLFormElement	CD:1546
HTMLFormElement.acceptCharset	CD:1547
HTMLFormElement.action	CD:1548
HTMLFormElement.elements	CD:1549
HTMLFormElement.enctype	CD:1549
HTMLFormElement.length	CD:1550
HTMLFormElement.method	CD:1551
HTMLFormElement.name	CD:1551
HTMLFormElement.reset()	CD:1552
HTMLFormElement.submit().....	CD:1553
HTMLFormElement.target	CD:1553
HTMLFrameElement	CD:1554
HTMLFrameElement.contentDocument	CD:1555
HTMLFrameElement.frameBorder	CD:1555
HTMLFrameElement.longDesc	CD:1556
HTMLFrameElement.marginHeight	CD:1557
HTMLFrameElement.marginWidth	CD:1557

HTMLFrameElement.name	CD:1558
HTMLFrameElement.noResize	CD:1558
HTMLFrameElement.scrolling.....	CD:1559
HTMLFrameElement.src	CD:1560
HTMLFrameSetElement.....	CD:1560
HTMLFrameSetElement.cols	CD:1561
HTMLFrameSetElement.rows	CD:1562
HTMLHeadElement	CD:1562
HTMLHeadElement.profile	CD:1563
HTMLHeadingElement	CD:1564
HTMLHeadingElement.align	CD:1564
HTMLHRElement	CD:1565
HTMLHRElement.align	CD:1566
HTMLHRElement.noShade	CD:1566
HTMLHRElement.size	CD:1567
HTMLHRElement.width	CD:1567
HTMLHtmlElement	CD:1568
HTMLHtmlElement.version	CD:1569
HTMLIFrameElement	CD:1569
HTMLIFrameElement.align	CD:1570
HTMLIFrameElement.contentDocument.....	CD:1571
HTMLIFrameElement.frameBorder.....	CD:1571
HTMLIFrameElement.height	CD:1572
HTMLIFrameElement.longDesc	CD:1573
HTMLIFrameElement.marginHeight.....	CD:1573
HTMLIFrameElement.marginWidth	CD:1574
HTMLIFrameElement.name.....	CD:1574
HTMLIFrameElement.scrolling	CD:1575
HTMLIFrameElement.src.....	CD:1576
HTMLIFrameElement.width	CD:1576
HTMLImageElement	CD:1577
HTMLImageElement.align.....	CD:1578
HTMLImageElement.alt.....	CD:1578
HTMLImageElement.border	CD:1579
HTMLImageElement.height.....	CD:1580
HTMLImageElement.hspace	CD:1580
HTMLImageElement.isMap.....	CD:1581
HTMLImageElement.longDesc	CD:1581
HTMLImageElement.lowSrc	CD:1582
HTMLImageElement.src	CD:1583
HTMLImageElement.useMap	CD:1583
HTMLImageElement.vspace	CD:1584
HTMLImageElement.width	CD:1584
HTMLInputElement	CD:1585
HTMLInputElement.accept	CD:1587

HTMLInputElement.accessKey	CD:1587
HTMLInputElement.align	CD:1588
HTMLInputElement.alt	CD:1588
HTMLInputElement.blur()	CD:1589
HTMLInputElement.checked	CD:1589
HTMLInputElement.click()	CD:1590
HTMLInputElement.defaultChecked	CD:1591
HTMLInputElement.defaultValue	CD:1591
HTMLInputElement.disabled	CD:1592
HTMLInputElement.focus()	CD:1593
HTMLInputElement.form	CD:1593
HTMLInputElement.maxLength	CD:1594
HTMLInputElement.name	CD:1594
HTMLInputElement.readOnly	CD:1595
HTMLInputElement.select()	CD:1596
HTMLInputElement.size	CD:1596
HTMLInputElement.src	CD:1597
HTMLInputElement.tabIndex	CD:1597
HTMLInputElement.type	CD:1598
HTMLInputElement.useMap	CD:1599
HTMLInputElement.value	CD:1599
HTMLIsIndexElement	CD:1600
HTMLIsIndexElement.form	CD:1601
HTMLIsIndexElement.prompt	CD:1601
HTMLLabelElement	CD:1602
HTMLLabelElement.accessKey	CD:1603
HTMLLabelElement.form	CD:1603
HTMLLabelElement.htmlFor	CD:1604
HTMLLegendElement	CD:1604
HTMLLegendElement.accessKey	CD:1605
HTMLLegendElement.align	CD:1606
HTMLLegendElement.form	CD:1606
HTMLLIElement	CD:1607
HTMLLIElement.type	CD:1608
HTMLLIElement.value	CD:1608
HTMLLinkElement	CD:1609
HTMLLinkElement.charset	CD:1610
HTMLLinkElement.disabled	CD:1610
HTMLLinkElement.href	CD:1611
HTMLLinkElement.hreflang	CD:1612
HTMLLinkElement.media	CD:1612
HTMLLinkElement.rel	CD:1613
HTMLLinkElement.rev	CD:1613
HTMLLinkElement.target	CD:1614
HTMLLinkElement.type	CD:1615

HTMLMapElement.....	CD:1615
HTMLMapElement.areas	CD:1616
HTMLMapElement.name	CD:1616
HTMLMenuElement.....	CD:1617
HTMLMenuElement.compact	CD:1618
HTMLMetaElement	CD:1618
HTMLMetaElement.content.....	CD:1619
HTMLMetaElement.httpEquiv	CD:1620
HTMLMetaElement.name	CD:1620
HTMLMetaElement.scheme.....	CD:1621
HTMLModElement	CD:1622
HTMLModElement.cite	CD:1622
HTMLModElement.dateTime	CD:1623
HTMLObjectElement	CD:1624
HTMLObjectElement.align	CD:1625
HTMLObjectElement.archive.....	CD:1625
HTMLObjectElement.border	CD:1626
HTMLObjectElement.code.....	CD:1627
HTMLObjectElement.codeBase.....	CD:1627
HTMLObjectElement.codeType.....	CD:1628
HTMLObjectElement.contentDocument	CD:1628
HTMLObjectElement.data	CD:1629
HTMLObjectElement.declare.....	CD:1630
HTMLObjectElement.form	CD:1630
HTMLObjectElement.height	CD:1631
HTMLObjectElement.hspace	CD:1631
HTMLObjectElement.name	CD:1632
HTMLObjectElement.standby	CD:1633
HTMLObjectElement.tabIndex	CD:1633
HTMLObjectElement.type	CD:1634
HTMLObjectElement.useMap	CD:1634
HTMLObjectElement.vspace	CD:1635
HTMLObjectElement.width	CD:1636
HTMLListElement.....	CD:1636
HTMLListElement.compact	CD:1637
HTMLListElement.start.....	CD:1637
HTMLListElement.type.....	CD:1638
HTMLOptGroupElement	CD:1639
HTMLOptGroupElement.disabled	CD:1639
HTMLOptGroupElement.label.....	CD:1640
HTMLOptionElement.....	CD:1641
HTMLOptionElement.defaultSelected	CD:1642
HTMLOptionElement.disabled	CD:1642
HTMLOptionElement.form	CD:1643
HTMLOptionElement.index	CD:1644

HTMLOptionElement.label	CD:1645
HTMLOptionElement.selected	CD:1645
HTMLOptionElement.text	CD:1646
HTMLOptionElement.value	CD:1647
HTMLParagraphElement	CD:1647
HTMLParagraphElement.align	CD:1648
HTMLParamElement	CD:1649
HTMLParamElement.name	CD:1649
HTMLParamElement.type	CD:1650
HTMLParamElement.value	CD:1651
HTMLParamElement.valueType	CD:1651
HTMLPreElement.....	CD:1652
HTMLPreElement.width	CD:1653
HTMLQuoteElement	CD:1653
HTMLQuoteElement.cite	CD:1654
HTMLScriptElement	CD:1654
HTMLScriptElement.charset	CD:1655
HTMLScriptElement.defer	CD:1656
HTMLScriptElement.event.....	CD:1657
HTMLScriptElement.htmlFor	CD:1657
HTMLScriptElement.src.....	CD:1657
HTMLScriptElement.text	CD:1658
HTMLScriptElement.type	CD:1658
HTMLSelectElement	CD:1659
HTMLSelectElement.add().....	CD:1660
HTMLSelectElement.blur()	CD:1661
HTMLSelectElement.disabled	CD:1661
HTMLSelectElement.focus()	CD:1662
HTMLSelectElement.form	CD:1663
HTMLSelectElement.length	CD:1663
HTMLSelectElement.multiple	CD:1664
HTMLSelectElement.name	CD:1665
HTMLSelectElement.options	CD:1665
HTMLSelectElement.remove().....	CD:1666
HTMLSelectElement.selectedIndex	CD:1667
HTMLSelectElement.size	CD:1667
HTMLSelectElement.tabIndex	CD:1668
HTMLSelectElement.type	CD:1669
HTMLSelectElement.value.....	CD:1669
HTMLStyleElement	CD:1670
HTMLStyleElement.disabled	CD:1671
HTMLStyleElement.media.....	CD:1671
HTMLStyleElement.type	CD:1672
HTMLTableCaptionElement.....	CD:1673
HTMLTableCaptionElement.align	CD:1673

HTMLTableCellElement.....	CD:1674
HTMLTableCellElement.abbr.....	CD:1675
HTMLTableCellElement.align	CD:1676
HTMLTableCellElement.axis	CD:1676
HTMLTableCellElement.bgColor.....	CD:1677
HTMLTableCellElement.cellIndex	CD:1677
HTMLTableCellElement.ch	CD:1678
HTMLTableCellElement.chOff.....	CD:1678
HTMLTableCellElement.colSpan	CD:1679
HTMLTableCellElement.headers	CD:1680
HTMLTableCellElement.height	CD:1680
HTMLTableCellElement.noWrap.....	CD:1681
HTMLTableCellElement.rowSpan	CD:1681
HTMLTableCellElement.scope.....	CD:1682
HTMLTableCellElement.vAlign.....	CD:1682
HTMLTableCellElement.width.....	CD:1683
HTMLTableColElement	CD:1683
HTMLTableColElement.align.....	CD:1684
HTMLTableColElement.ch.....	CD:1685
HTMLTableColElement.chOff	CD:1686
HTMLTableColElement.span	CD:1686
HTMLTableColElement.vAlign	CD:1687
HTMLTableColElement.width	CD:1687
HTMLTableElement	CD:1688
HTMLTableElement.align	CD:1689
HTMLTableElement.bgColor	CD:1690
HTMLTableElement.border	CD:1690
HTMLTableElement.caption.....	CD:1691
HTMLTableElement.cellPadding	CD:1692
HTMLTableElement.cellSpacing	CD:1692
HTMLTableElement.createCaption()	CD:1693
HTMLTableElement.createTFoot()	CD:1693
HTMLTableElement.createTHead()	CD:1694
HTMLTableElement.deleteCaption()	CD:1695
HTMLTableElement.deleteRow().....	CD:1695
HTMLTableElement.deleteTFoot()	CD:1696
HTMLTableElement.deleteTHead()	CD:1696
HTMLTableElement.frame	CD:1697
HTMLTableElement.insertRow()	CD:1698
HTMLTableElement.rows.....	CD:1698
HTMLTableElement.rules.....	CD:1699
HTMLTableElement.summary	CD:1699
HTMLTableElement.tBodies	CD:1700
HTMLTableElement.tFoot	CD:1701
HTMLTableElement.tHead.....	CD:1701

HTMLTableElement.width	CD:1702
HTMLTableRowElement	CD:1702
HTMLTableRowElement.align	CD:1703
HTMLTableRowElement.bgColor	CD:1704
HTMLTableRowElement.cells	CD:1705
HTMLTableRowElement.ch	CD:1705
HTMLTableRowElement.chOff	CD:1706
HTMLTableRowElement.deleteCell()	CD:1706
HTMLTableRowElement.insertCell().....	CD:1707
HTMLTableRowElement.rowIndex	CD:1708
HTMLTableRowElement.sectionRowIndex	CD:1708
HTMLTableRowElement.vAlign	CD:1709
HTMLTableSectionElement	CD:1709
HTMLTableSectionElement.align	CD:1710
HTMLTableSectionElement.ch.....	CD:1711
HTMLTableSectionElement.chOff	CD:1711
HTMLTableSectionElement.deleteRow().....	CD:1712
HTMLTableSectionElement.insertRow()	CD:1713
HTMLTableSectionElement.rows.....	CD:1713
HTMLTableSectionElement.vAlign	CD:1714
HTMLTextAreaElement	CD:1714
HTMLTextAreaElement.accessKey	CD:1715
HTMLTextAreaElement.blur()	CD:1716
HTMLTextAreaElement.cols	CD:1717
HTMLTextAreaElement.defaultValue	CD:1717
HTMLTextAreaElement.disabled	CD:1718
HTMLTextAreaElement.focus()	CD:1718
HTMLTextAreaElement.form.....	CD:1719
HTMLTextAreaElement.name	CD:1720
HTMLTextAreaElement.readOnly	CD:1720
HTMLTextAreaElement.rows.....	CD:1721
HTMLTextAreaElement.select().....	CD:1721
HTMLTextAreaElement.tabIndex.....	CD:1722
HTMLTextAreaElement.type	CD:1723
HTMLTextAreaElement.value	CD:1723
HTMLTitleElement.....	CD:1724
HTMLTitleElement.text	CD:1725
HTMLUListElement.....	CD:1725
HTMLUListElement.compact	CD:1726
HTMLUListElement.type.....	CD:1727

12 DOM EVENTS	(CD-ROM) 1731
DocumentEvent.....	CD:1731
DocumentEvent.createEvent()	CD:1732
Event	CD:1732

Event.type	CD:1734
Event.bubbles	CD:1734
Event.cancelable	CD:1735
Event.currentTarget	CD:1736
Event.eventPhase	CD:1736
Event.initEvent()	CD:1737
Event.preventDefault()	CD:1738
Event.stopPropagation()	CD:1739
Event.target	CD:1739
Event.timeStamp	CD:1740
EventException	CD:1741
EventException.code	CD:1742
EventListener	CD:1743
EventTarget	CD:1744
EventTarget.addEventListener()	CD:1745
EventTarget.dispatchEvent()	CD:1746
EventTarget.removeEventListener()	CD:1747
MouseEvent	CD:1748
MouseEvent.altKey	CD:1749
MouseEvent.button	CD:1749
MouseEvent.clientX	CD:1750
MouseEvent.clientY	CD:1751
MouseEvent.ctrlKey	CD:1751
MouseEvent.initMouseEvent()	CD:1752
MouseEvent.metaKey	CD:1753
MouseEvent.relatedTarget	CD:1753
MouseEvent.screenX	CD:1754
MouseEvent.screenY	CD:1754
MouseEvent.shiftKey	CD:1755
MutationEvent	CD:1756
MutationEvent.relatedNode	CD:1757
MutationEvent.attrChange	CD:1758
MutationEvent.attrName	CD:1758
MutationEvent.initMutationEvent()	CD:1759
MutationEvent.newValue	CD:1760
MutationEvent.prevValue	CD:1760
UIEvent	CD:1761
UIEvent.detail	CD:1762
UIEvent.initUIEvent()	CD:1762
UIEvent.view	CD:1763

13 DOM CSS

(CD-ROM) 1767

Counter	CD:1767
Counter.identifier	CD:1768
Counter.listStyle	CD:1768

Counter.separator	CD:1769
CSSCharsetRule	CD:1770
CSSCharsetRule.encoding	CD:1770
CSSFontFaceRule	CD:1771
CSSFontFaceRule.style.....	CD:1771
CSSImportRule	CD:1772
CSSImportRule.href	CD:1773
CSSImportRule.media	CD:1773
CSSImportRule.styleSheet	CD:1774
CSSMediaRule	CD:1774
CSSMediaRule.cssRules.....	CD:1775
CSSMediaRule.deleteRule()	CD:1776
CSSMediaRule.insertRule()	CD:1776
CSSMediaRule.media	CD:1777
CSSPageRule	CD:1778
CSSPageRule.selectorText	CD:1778
CSSPageRule.style	CD:1779
CSSPrimitiveValue	CD:1779
CSSPrimitiveValue.getCounterValue()	CD:1782
CSSPrimitiveValue.getFloatValue()	CD:1782
CSSPrimitiveValue.getRectValue().....	CD:1783
CSSPrimitiveValue.getRGBColorValue().....	CD:1783
CSSPrimitiveValue.getStringValue()	CD:1784
CSSPrimitiveValue.primitiveType	CD:1785
CSSPrimitiveValue.setFloatValue()	CD:1785
CSSPrimitiveValue.setStringValue().....	CD:1786
CSSRule	CD:1786
CSSRule.cssText	CD:1787
CSSRule.parentRule	CD:1788
CSSRule.parentStyleSheet	CD:1789
CSSRule.type	CD:1789
CSSRuleList	CD:1790
CSSRuleList.item()	CD:1791
CSSRuleList.length.....	CD:1791
CSSStyleDeclaration.....	CD:1792
CSSStyleDeclaration.cssText	CD:1793
CSSStyleDeclaration.getPropertyCSSValue()	CD:1794
CSSStyleDeclaration.getPropertyPriority()	CD:1794
CSSStyleDeclaration.getPropertyValue()	CD:1795
CSSStyleDeclaration.item()	CD:1796
CSSStyleDeclaration.length	CD:1796
CSSStyleDeclaration.parentRule	CD:1797
CSSStyleDeclaration.removeProperty()	CD:1797
CSSStyleDeclaration.setProperty().....	CD:1798
CSSStyleRule	CD:1799

CSSStyleRule.selectorText	CD:1799
CSSStyleRule.style	CD:1800
CSSStyleSheet	CD:1800
CSSStyleSheet.cssRules	CD:1801
CSSStyleSheet.deleteRule()	CD:1802
CSSStyleSheet.insertRule()	CD:1802
CSSStyleSheet.ownerRule	CD:1803
CSSUnknownRule	CD:1804
CSSValue	CD:1804
CSSValue.cssText	CD:1805
CSSValue.cssValueType	CD:1806
CSSValueList	CD:1806
CSSValueList.item()	CD:1807
CSSValueList.length	CD:1808
DocumentCSS	CD:1809
DocumentCSS.getOverrideStyle()	CD:1809
DOMImplementationCSS	CD:1810
DOMImplementationCSS.createCSSStyleSheet()	CD:1810
ElementCSSInlineStyle	CD:1811
ElementCSSInlineStyle.style	CD:1812
Rect	CD:1812
Rect.bottom	CD:1813
Rect.left	CD:1814
Rect.right	CD:1814
Rect.top	CD:1815
RGBColor	CD:1815
RGBColor.blue	CD:1816
RGBColor.green	CD:1817
RGBColor.red	CD:1817
ViewCSS	CD:1818
ViewCSS.getComputedStyle()	CD:1818

14 JSCRIPT RUNTIME

(CD-ROM) 1823

Dictionary	CD:1824
Dictionary.Add()	CD:1825
Dictionary.Count	CD:1826
Dictionary.Exists()	CD:1826
Dictionary.Item()	CD:1827
Dictionary.Items()	CD:1828
Dictionary.Key()	CD:1829
Dictionary.Keys()	CD:1830
Dictionary.Remove()	CD:1831
Dictionary.RemoveAll()	CD:1832
Drive	CD:1832
Drive.AvailableSpace	CD:1834

Drive.DriveLetter	CD:1834
Drive.DriveType	CD:1835
Drive.FileSystem.....	CD:1836
Drive.FreeSpace	CD:1837
Drive.IsReady	CD:1838
Drive.Path	CD:1838
Drive.RootFolder	CD:1839
Drive.SerialNumber	CD:1840
Drive.ShareName	CD:1840
Drive.TotalSize	CD:1841
Drive.VolumeName.....	CD:1842
Drives	CD:1842
Drives.Count	CD:1843
Drives.Item()	CD:1844
File	CD:1845
File.Attributes	CD:1846
File.Copy()	CD:1847
File.DateCreated	CD:1848
File.DateLastAccessed	CD:1849
File.DateLastModified	CD:1850
File.Delete.....	CD:1851
File.Drive	CD:1852
File.Move()	CD:1853
File.Name	CD:1854
File.OpenAsTextStream()	CD:1854
File.ParentFolder.....	CD:1856
File.Path	CD:1857
File.ShortName	CD:1857
File.ShortPath	CD:1858
File.Size	CD:1859
File.Type	CD:1860
Files.....	CD:1861
Files.Count	CD:1862
Files.Item	CD:1863
FileSystemObject	CD:1864
FileSystemObject.BuildPath()	CD:1865
FileSystemObject.CopyFile	CD:1866
FileSystemObject.CopyFolder()	CD:1867
FileSystemObject.CreateFolder()	CD:1869
FileSystemObject.CreateTextFile().....	CD:1869
FileSystemObject.DeleteFile()	CD:1871
FileSystemObject.DeleteFolder()	CD:1872
FileSystemObject.DriveExists()	CD:1873
FileSystemObject.Drives	CD:1874
FileSystemObject.FileExists()	CD:1874

FileSystemObject.FolderExists()	CD:1876
FileSystemObject.GetAbsolutePathName()	CD:1877
FileSystemObject.GetBaseName()	CD:1878
FileSystemObject.GetDrive()	CD:1879
FileSystemObject.GetDriveName()	CD:1879
FileSystemObject.GetExtensionName()	CD:1880
FileSystemObject.GetFile()	CD:1882
FileSystemObject.GetFileName()	CD:1882
FileSystemObject.GetFolder()	CD:1883
FileSystemObject.GetParentFolderName()	CD:1885
FileSystemObject.GetSpecialFolder()	CD:1885
FileSystemObject.GetTempName()	CD:1886
FileSystemObject.MoveFile()	CD:1887
FileSystemObject.MoveFolder()	CD:1888
FileSystemObject.OpenTextFile()	CD:1889
Folder	CD:1891
Folder.Attributes	CD:1892
Folder.Copy()	CD:1893
Folder.DateCreated	CD:1894
Folder.DateLastAccessed	CD:1895
Folder.DateLastModified	CD:1896
Folder.Delete()	CD:1897
Folder.Drive	CD:1898
Folder.Files	CD:1899
Folder.IsRootFolder	CD:1899
Folder.Move()	CD:1900
Folder.Name	CD:1901
Folder.ParentFolder	CD:1902
Folder.Path	CD:1903
Folder.ShortName	CD:1903
Folder.ShortPath	CD:1904
Folder.Size	CD:1905
Folder.SubFolders	CD:1906
Folder.Type	CD:1907
Folders	CD:1908
Folders.Add()	CD:1909
Folders.Count	CD:1910
Folders.Item	CD:1911
TextStream	CD:1912
TextStream.AtEndOfLine	CD:1913
TextStream.AtEndOfStream	CD:1914
TextStream.Close()	CD:1915
TextStream.Column	CD:1915
TextStream.Line	CD:1916
TextStream.Read()	CD:1917

TextStream.ReadAll()	CD:1918
TextStream.ReadLine()	CD:1919
TextStream.Skip()	CD:1919
TextStream.SkipLine()	CD:1920
TextStream.Write()	CD:1921
TextStream.WriteBlankLines()	CD:1922
TextStream.WriteLine()	CD:1923

15 WINDOWS SCRIPT HOST (CD-ROM) 1927

Scripting.Signer	CD:1927
WScript	CD:1929
WScript.Arguments	CD:1930
WScript.ConnectObject()	CD:1931
WScript.CreateObject()	CD:1932
WScript.DisconnectObject()	CD:1933
WScript.Echo()	CD:1933
WScript.FullName	CD:1934
WScript.GetObject()	CD:1934
WScript.Interactive	CD:1935
WScript.Name	CD:1936
WScript.Path	CD:1937
WScript.Quit()	CD:1937
WScript.ScriptFullName	CD:1938
WScript.ScriptName	CD:1938
WScript.Sleep()	CD:1939
WScript.StdErr	CD:1939
WScript.StdIn	CD:1940
WScript.StdOut	CD:1941
WScript.Version	CD:1942
WScript.WshArguments	CD:1942
WScript.WshController	CD:1943
WScript.WshNetwork	CD:1943
WScript.WshShell	CD:1944
WshArguments	CD:1944
WshArguments.Count	CD:1945
WshArguments.Count()	CD:1946
WshArguments.Item()	CD:1946
WshArguments.length	CD:1947
WshArguments.Named	CD:1948
WshArguments.ShowUsage()	CD:1949
WshArguments.Unnamed	CD:1950
WshArguments.WshNamed	CD:1951
WshArguments.WshUnnamed	CD:1952
WshController	CD:1952
WshController.CreateScript()	CD:1953

WshController.WshRemote	CD:1954
WshEnvironment	CD:1954
WshEnvironment.Count()	CD:1956
WshEnvironment.Item()	CD:1957
WshEnvironment.length	CD:1957
WshEnvironment.Remove()	CD:1958
WshNamed	CD:1960
WshNamed.Count()	CD:1961
WshNamed.Exists()	CD:1961
WshNamed.Item	CD:1962
WshNamed.length.....	CD:1963
WshNetwork	CD:1963
WshNetwork.AddPrinterConnection()	CD:1964
WshNetwork.AddWindowsPrinterConnection()	CD:1965
WshNetwork.ComputerName.....	CD:1966
WshNetwork.EnumNetworkDrives()	CD:1967
WshNetwork.EnumPrinterConnection().....	CD:1967
WshNetwork.MapNetworkDrive()	CD:1968
WshNetwork.RemoveNetworkDrive()	CD:1969
WshNetwork.RemovePrinterConnection()	CD:1970
WshNetwork.SetDefaultPrinter()	CD:1970
WshNetwork.UserDomain	CD:1971
WshNetwork.UserName	CD:1971
WshRemote.....	CD:1972
WshRemote.End	CD:1973
WshRemote.Error (Event)	CD:1974
WshRemote.Error (Property)	CD:1974
WshRemote.Execute()	CD:1975
WshRemote.WshRemoteError	CD:1976
WshRemote.Start	CD:1976
WshRemote.Status	CD:1977
WshRemote.Terminate()	CD:1978
WshRemoteError	CD:1979
WshRemoteError.Character	CD:1980
WshRemoteError.Description.....	CD:1980
WshRemoteError.Line	CD:1981
WshRemoteError.Number	CD:1982
WshRemoteError.Source	CD:1983
WshRemoteError.SourceText	CD:1984
WshScriptExec	CD:1985
WshScriptExec.Status.....	CD:1985
WshScriptExec.StdErr	CD:1986
WshScriptExec.StdIn	CD:1987
WshScriptExec.StdOut	CD:1988
WshScriptExec.Terminate()	CD:1989

WshShell	CD:1990
WshShell.AppActivate()	CD:1991
WshShell.CreateShortcut()	CD:1991
WshShell.CurrentDirectory	CD:1992
WshShell.Environment	CD:1993
WshShell.Exec()	CD:1993
WshShell.ExpandEnvironmentStrings()	CD:1994
WshShell.LogEvent()	CD:1995
WshShell.Popup()	CD:1996
WshShell.RegDelete()	CD:1997
WshShell.RegRead()	CD:1998
WshShell.RegWrite()	CD:1998
WshShell.Run()	CD:1999
WshShell.SendKeys()	CD:2000
WshShell.SpecialFolders	CD:2002
WshShell.WshShortcut	CD:2003
WshShell.WshUrlShortcut	CD:2003
WshShell.WshEnvironment	CD:2003
WshShell.WshSpecialFolders	CD:2004
WshShell.WshScriptExec	CD:2004
WshShortcut	CD:2004
WshShortcut.Arguments	CD:2005
WshShortcut.Description	CD:2006
WshShortcut.FullName	CD:2007
WshShortcut.Hotkey	CD:2007
WshShortcut.IconLocation	CD:2008
WshShortcut.Save()	CD:2009
WshShortcut.TargetPath	CD:2010
WshShortcut.WindowStyle	CD:2010
WshShortcut.WorkingDirectory	CD:2011
WshSpecialFolders	CD:2012
WshSpecialFolders.Count	CD:2013
WshSpecialFolders.Item	CD:2014
WshSpecialFolders.length	CD:2015
WshUnnamed	CD:2015
WshUnnamed.Count()	CD:2017
WshUnnamed.Item	CD:2018
WshUnnamed.length	CD:2019
WshUrlShortcut	CD:2019
WshUrlShortcut.FullName	CD:2020
WshUrlShortcut.Save()	CD:2021
WshUrlShortcut.TargetPath	CD:2022

About the Authors

R. Allen Wyke, of Durham, North Carolina is vice president of Research and Development at the eMarketing solutions company Engage, where he works with product managers, product marketing, and engineering to help ensure that their products have the proper vision and direction in both online and offline worlds. He has also developed intranet Web pages for a leading telecommunications and networking company, as well as worked on several Internet sites.

Wyke is the author of seven books on various Internet technologies including Perl, JavaScript, and PHP. In the past, he has also written the monthly *Webmaster* column for SunWorld, and a weekly article, "Integrating Windows and Unix," for ITworld.com.

Jason D. Gilliam is a software developer at ichat, Inc. At ichat, he develops Web-based chat and discussion group software using C++, JavaScript, HTML, and various other Internet technologies. In addition to his work at ichat, he has developed intranet Web pages and numerous C++ applications for a leading telecommunications company and has worked on several Internet sites for other organizations. Gilliam is also a contributing author to *JavaScript Unleashed, Third Edition*. He holds a bachelor's degree in Computer Engineering from North Carolina State University. Jason and his wife live in Cary, North Carolina.

Charlton Ting, of Vienna, Virginia is a software engineer for KOZ Inc who began programming in JavaScript with its first beta release. He has worked extensively with many Internet technologies while developing Internet telephony solutions at Lucent Technologies. His programming experience includes a variety of languages such as Java, C++, Smalltalk, Perl, HTML, XML, and Pascal. Ting holds degrees in Computer Engineering and Electrical Engineering from North Carolina State University.

Sean Michaels is a software engineer at Avesair, Inc. where he designs and implements next generation mobile commerce applications. He has been in the Internet space since 1994 and has a depth of experience in developing highly distributed fault tolerant applications. He has used Java, C++, and XML extensively over the years. Previously, he coauthored the *Official Netscape ONE* book and the *Official Netscape Technologies Developer Guide* from Netscape Press. Michaels holds a degree in Latin from Millsaps College in Jackson, MS and a degree in Computer Science from Mississippi College.

Dedications

This one is for my sisters: Sandra, Valerie, and Evelyn. They truly are inspirations in my life and have broadened my visions and provided support well beyond anything I would have ever imagined. I love each of you very much.

—R. Allen Wyke

I dedicate this book to my wonderful wife, Deena, who is the greatest! Your constant encouragement, love, and support has carried me through the writing of this book. I love you!

—Jason D. Gilliam

This book is dedicated to my family: John, Alice, Angela, Melissa, Olivia, and Tilly. Thank you for all your love and support for everything I've done in my life. I love you all very much.

—Charlton Ting

I would like to dedicate this book to my wife April and my two children Noah, and Anna, and to the memory of my father Alan Michaels without whose influence I would not be here today.

—Sean Michaels

Acknowledgments

R. Allen Wyke

On the publishing side, I would like to thank Bob Kern of TIPS Publishing and my coauthors, Jason and Chuck, for their professionalism, hard work, and overall support in the proposing and writing of this book. I would also like to thank Shelley Johnston-Markanday, who has been nothing short of an absolutely fabulous acquisitions editor, and Jon Steever, who developed the book and kept us focused. Additionally, I would like to thank everyone at Sams who worked on the book and helped make sure that was the best it could be.

I would also like to thank Dan Jaye for the professional opportunities he has allowed me to pursue at Engage, as well as the pressure to push the envelope a little more. And finally, I would like to thank the wonderful woman in my life, J, and the rest of the “Raccoons.” It’s never a dull moment around any of you.

Jason D. Gilliam

I would like to thank Bob Kern of TIPS Publishing and my coauthors, Allen and Chuck, for their efforts in developing and writing this book. I would also like to thank our acquisitions editor, Shelley Johnston-Markanday, and development editor, Jon Steever, for all their hard work as well as everyone at Sams who helped make this book a success.

I would also like to thank the “lunch guys” for their open ears and words of encouragement.

Charlton Ting

I would like to thank Bob Kern of TIPS Publishing for all his hard work in making this book possible. I also want to thank my coauthors, Allen and Jason, for all their hard work, dedication, and encouragement to bring this book together and make it such a great success. It's truly an honor to work with you guys. Additionally I would like to thank Shelley Johnston-Markanday, Scott Meyers, Jon Steever, and everyone at Sams who worked so hard at making this book so successful.

I also want to thank all my friends who have been there when I needed them: Mike, Carolyn, Monty, Theresa, Mitch, Sabrina, John O, Sunil "the Z Doc" Cherukuri, Blanke, Cosima, Nairn, Airmer Roberts, and anyone I may have forgotten to mention. You guys are the greatest friends anyone could have.

Sean Michaels

I would like to thank Bob Kern at TIPS Publishing for putting up with my antics during the authoring process. Also, the coauthors: Allen, Chuck, and Jason for giving me a solid foundation to work with and making the first edition of this book successful. I would also like to thank the folks at Sams Publishing: Shelley Johnston-Markanday, Scott Meyers, and Jon Steever for guiding the book through the process.

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can e-mail or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax: 317-581-4770

Email: webdev@sampublishing.com

Mail: Mark Taber
Associate Publisher
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Introduction

Welcome to *Pure JavaScript, 2nd Edition*! This book has been written by JavaScript programmers for JavaScript programmers, and has been updated to cover the most current version of the language. It is your complete reference for developing, testing, and deploying JavaScript solutions in pages and on Web sites.

Pure JavaScript was not written to teach a person how to program, but rather to provide the details and semantics of the JavaScript language so programmers can exploit it as they see fit. JavaScript has evolved over the past few years and is reaching into new areas, especially in the most current browsers and server-side environments, most of which are addressed in this book. The book itself is broken into three main parts, including references on concepts, techniques, and syntax. Each represents a valuable step in learning and using the language.

Part I, “A Programmer’s Overview of JavaScript,” acts as a bridge for programmers who are currently programming in another language. It covers some programming techniques, such as the use of regular expressions, event handlers, exceptions, and even debugging. Many times programmers don’t want to buy a beginners book on JavaScript because they do not need to know how to program, but rather they need specifics about the language. When they know these specifics and semantics, the syntax is easy. This section provides the necessary information for such a programming migration.

Part II, “Programming in JavaScript,” shows you the advantages and strengths of JavaScript. It discusses some of the pros and cons of using the language on the client-side, server-side, and even within the Windows Script Host environment. Programmers will learn how to use JavaScript in real-world instances. They will be introduced to some of the browser issues as well as how to process Web information. In addition, programmers will be shown how to access Java functions within an applet and to use server-side JavaScript for Internet, intranet, or extranet development. After you have completed this section, you will be ready to move forward and start programming. This leads you into the last section of the book.

Part III, “JavaScript Reference,” makes up the majority of the book and contains some of the most useful information for current JavaScript programmers—reference material organized by object. Each property, method, and event is discussed in detail under its associated object; and you’ll see an example of its use. Each entry also shows the appropriate language version and environment (browser, server, and so on) support.

The section itself is broken into nine chapters. The first chapter covers the core JavaScript objects and syntax. The next chapter covers objects specific to the client-side, which is where JavaScript really has its roots. The third, fourth, fifth, and sixth chapter in this section outlines the various support for the Document Object Model (DOM) within JavaScript. The seventh chapter covers server-side JavaScript objects for the Netscape and iPlanet Enterprise servers, and the Active Server pages (ASP) environment present in Microsoft’s IIS Web Servers. The second to last chapter in the section covers the JScript RunTime objects, and the final chapter covers Windows Script Host.

2 Introduction

And that covers it! For new JavaScript programmers, welcome to the world of JavaScript. For those of you wanting a good, solid reference for your programming needs, we hope you find this book to be the most resourceful and current title on the shelves today!

R. Allen Wyke
Jason Gilliam
Charlton Ting
Sean Michaels



PART I

A PROGRAMMER'S OVERVIEW OF JAVASCRIPT

- 1 What Is JavaScript to a Programmer? 5
- 2 Details of the Language 31



CHAPTER 1

What Is JavaScript to a Programmer?

In the beginning, there were Assembly and compiled languages. Later came scripting languages such as sed, awk, and Perl, which many programmers used to perform a variety of tasks. Followed by, in the late 80s and early 90s, the Internet, which exploded into a technological revolution that allowed anyone with a modem to communicate and retrieve information from around the world. As the Internet grew in number of users, it was obvious that an increase in functionality was needed in the browsers and the data they were rendering.

HTML, even with its advantages, was falling short of providing the control many developers wanted when creating Web pages and applications. This prompted the use of server-side programs, or scripts as they were often called, to handle some of the page dynamics developers needed from their sites.

These programs helped Web developers by allowing them to increase a site's functionality as well as process user-submitted information. However, *CGI*, or *common gateway interface*, scripts had to generate and return a response when the user sent incorrect or incomplete information. This led to the unnecessary back-and-forth transmission of data between browser and server. But, overall, it was a minor price to pay for the functionality it provided.

With time, and an increase in traffic, it became increasingly obvious that client-side intelligence was needed to offload some of the CGI functionality. Something was needed to perform this error checking and to decrease the amount of time a user spent connecting to a server to validate data. This would

also enable the Web site to offload some of its processing load to the browser machine, which meant an increase in the overall performance of a site.

It was partially this lack of client-side functionality and efficiency that helped spawn a new scripting language—one that could be executed within a browser's environment and not on the server. This language could be used to perform client-side tasks such as form validation and dynamic page content creation—one that would put the programming into HTML publishing. Welcome to the birth of JavaScript.

Welcome to JavaScript

On December 4, 1995, Netscape and Sun jointly introduced JavaScript 1.0, originally called LiveScript, to the world. This language, unlike its server-based predecessors, could be interpreted within the then new Netscape Navigator 2 browsers. As an interpreted language, JavaScript was positioned as a complement to Java and would allow Web developers to create and deploy custom applications across the enterprise and Internet alike. JavaScript gave Web developers the power to truly program—not just format data with HTML.

In addition to the client-side control developers desired, Netscape implemented server-side JavaScript. This allowed developers to use the same programming language on the server as they did in their pages for browsers. Database connection enhancements were added to the language (called LiveWire), allowing the developer to pull information directly from a database and maintain user sessions for common functionality such as shopping carts. JavaScript had truly bridged the gap between the simple world of HTML and the more complex CGI programs on the server. It provided a common language for Web developers to design, implement, and deploy solutions across their networks and distributed the overall processing load of their applications.

The next level of acceptance in the world of JavaScript was Microsoft's implementation of the language in its Internet Explorer 3 browser—the implementation was called JScript. Similar to Netscape, Microsoft also implemented the language on the server-side (JScript 2.0) within its ASP (Active Server Pages) environment. It also allowed developers the flexibility of using a common language on both the client and server-side, while providing many of the robust features, such as object invocation and usage, in compiled languages.

JAVASCRIPT VERSUS JSCRIPT, AND WHAT IS ECMASCRIPT?

JScript 1.0 was based on the published documentation from Netscape, so essentially it is the same thing as JavaScript 1.0. However, there were a few "features" that Netscape did not publish, as well as some functionality that was not re-created by Microsoft correctly. The result of this is that there are some discrepancies between JScript 1.0 and JavaScript 1.0.

Since the release of these initial browsers, JavaScript and JScript were both submitted to the ECMA (European Computer Manufacturers Association) standardization

body and have become the standard known as ECMAScript (ECMA-262). Because of this standardization, it is now considered that JavaScript is Netscape's implementation of ECMAScript while JScript is Microsoft's implementation.

The adoption of the first edition of ECMAScript occurred in June 1997 followed by its adoption by the International Organization for Standardization and International Electrotechnical Commission in April 1998 (ISO/IEC 16262). A second edition of the standard was approved by ECMA in June 1998, and a third edition was adopted in December 1999.



NOTE

Because Netscape's JavaScript was the foundation of all this, the book will refer to JavaScript, JScript, and ECMAScript simply as JavaScript except where a differentiation is needed.

So, what is JavaScript to the programmer? Well, in its purest form, it is an object-based, cross-platform, loosely-typed, multi-use language that allows a programmer to deploy many types of solutions to many clients. It not only involves adding functionality to Web pages as rendered within a browser, it also allows server-side processing for Netscape and Microsoft Web servers.

JScript has also been included in Microsoft's Windows Script Host (WSH), to allow programmers to write scripts to be executed on the operating system itself, and most recently as a major language under their .NET strategy (more on that later). When operating within the WSH environment, JScript is similar to the old DOS batch files, but gives programmers more functionality and versatility in what they can accomplish. This type of advancement has allowed the language to take hold in the computer world and continue to progress.

In addition to the benefits of these environments in which JavaScript can be executed, security measures are in place to protect end users against malicious code. Even though it is still young in terms of age, JavaScript is very mature and powerful. This functionality, ability, and versatility positions JavaScript as the best solution for many programmers.

Now that you've learned about what JavaScript is, you should dive a little deeper into what it means to a programmer. Being programmers ourselves, we know that a few strategically placed words do not make a language useful; so first, we'll look at the object-based characteristics of JavaScript.

Object-Based Technology

The fact that you are reading this reference somewhat implies that you have programmed in JavaScript or at least one other language before, even if only for one semester in college. Going one step further, I bet the language you programmed in was either C++, Java, or Perl—with each having various levels of object orientation (OO). Java specifically is OO by virtue of having all programmer created objects extend from core Java language classes or their own.

Object-Oriented Programming

For those of you unfamiliar with object-oriented programming (OOP), it is a concept that allows you to create reusable objects or classes in code. An object or class has associated with it various characteristics and functionality that defines what kind of properties and states it can take on. After these are created and defined, it is possible to create new instances—sometimes referred to as *children*—that inherit the capability to have the same characteristics of their *parent* object.

To give you an example of how this might work, let's create a vehicle object. Some of the characteristics assigned to this vehicle object are the number of doors, the color, and the type (such as sports car or truck). In addition to these characteristics, let's define the ability to move or stop the vehicle. The pseudo-code for this type of object might look something similar to the following:

```
object vehicle(){  
    // Characteristics of the vehicle  
    num_doors;  
    color;  
    type;  
  
    // Methods used to move and stop the truck. Note that the move()  
    // method takes a direction as an argument. This direction could  
    // be something like forward, backward, left, or right.  
    move(direction);  
    stop();  
}
```

Now that this vehicle object is defined, it is easy to create new instances of it. A vehicle that is a car with four doors and is red can be easily created. You could also create a vehicle that is a truck with two doors and is black. The possibilities are endless.

In addition to creating these instances of the vehicle object, you have also made it possible to program in the ability to change the state of your instance. This is accomplished by specifying whether it is stopped or moving. Again, this method of programming can make the possibilities endless and certainly reusable.

Here is another example in pseudo-code to illustrate this concept. The sample creates a black, two-door truck that is moving forward:

```
// Create the new instance of the vehicle  
myTruck = new vehicle();  
  
// Define the type, number of doors and color  
myTruck.doors = 2;  
myTruck.color = "black";  
myTruck.type = "truck";  
  
// Define the "state" of the truck  
myTruck.move(forward);
```

The basic process here is to create an instance of the vehicle object and then to assign characteristic values to it. It is these values that make it a unique instance of this object, which we have specified as a truck.

The existence of the vehicle object itself allows us to easily create more vehicles with different characteristics. When programming, this “ease” translates into less code—something all programmers like to hear.

Now that this object is defined, it is possible to create new instances that inherit its characteristics without having to redefine them. You are able to capitalize on any overlaps in characteristics within objects by doing this. The idea is to create a general, master object that gives you the ability to then derive child instances that provide all the functionality and characteristics you need.

We can take this a step further by creating new objects—not instances—that inherit the parent objects’ characteristics. Doing so allows us to derive child instances from the child object that we have decided will inherit only certain characteristics. We could define a child object to only pass on the parent object’s color characteristic to any child instances of its own. It is the concept of this object orientation that allows you to perform this modular type of programming.

The following pseudo-code example shows how we could create an airplane object based on the previous vehicle object:

```
// Create the new object that inherits the vehicle
object airplane(){

    // Inherit the vehicle object
    this = new vehicle();

    // Define the doors property, then assign it to the size
    // property of the plane object, which makes the most sense
    this.doors = "747";
    this.size = this.doors;

    // Assign the color and type of plane
    this.color = "silver";
    this.type = "American Airlines";

    // Define the "state" of the plane
    this.move(up);

    // Now that the object is created with the values, return the
    // object.
    return this;
}
```

Not all languages support this concept, and there are other languages only based on its concepts. This concept definitely supplies advantages to the language, but it is not required to write good, effective, modular code. JavaScript is a perfect example of how

a language has applied some of these concepts, but is not completely OO. It does this by being *object based*.

NOTE

Talking about OOP in further detail is beyond the focus of a JavaScript book, but it is worth some investigation if you are a real programming enthusiast. Check out your local bookstore for a selection of titles on this subject. You can also visit Object Central (<http://www.objectcentral.com>) on the Web for reference and links to OOP information.

So how does object-based programming fit into the equation? It is very similar to OO except that it does not have all the functionality or characteristics. There are limited amounts of inheritance, scope, and functionality that you can perform with an object-based language. This should not be taken as mark against JavaScript, because it makes the language easier to learn and maintain for the developer. OOP is no easy beast to tackle and will provide many headaches before it is implemented correctly.

JavaScript also makes up for many of its OO limitations by allowing you to create your own object-like elements, as well as extend the core objects in the language by prototyping new properties. To get an idea of how this is done, take a look at JavaScript object orientation.

Object Orientation of JavaScript

Before we go into a lot of detail on the object orientation of JavaScript, let's outline the details of the core components as well as between server-side and client-side objects. Both sets of objects are specific to their runtime environment, so default object initialization and creation occur at different times. Because of this characteristic, you will look at the language in several parts:

- Core
- Client-side
- Server-side
- JScript-specific
- Windows Script Host

Core

First and foremost, it is important for you to know and understand the core objects in the JavaScript language. These objects are generally found across all implementations and are defined in the ECMAScript standard. These objects lay the foundation for the shared functionality, such as mathematical, array, or date related, which are used in most all scripts. Figure 1.1 shows a list of these core objects.

As you can see, the core objects are Array, Boolean, Date, Function, Global, Math, Number, Object, RegExp, Error, and String. In addition to these objects, both Netscape

and Microsoft have created objects specific to their core implementations. These are not specific to any environment, and are therefore core objects in the sense of their consistency. Figure 1.2 shows these objects.

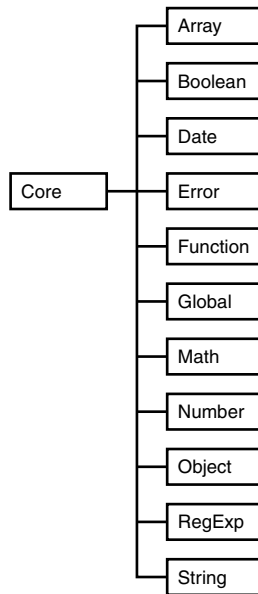


Figure 1.1

Core ECMAScript object hierarchy.

Client-Side

Client-side JavaScript is, at its lowest level, a set of objects created when a page is loaded in the browser. In addition, there are also objects that revolve around the browser loading the page and other derived objects that are created when certain tags are contained on a page. These derived objects inherit some of the various characteristics of their parent object and also allow scripting access to the HTML tag's properties.

Understanding the hierarchy of the JavaScript objects is essential if you plan on doing any in-depth programming. You will get a better understanding of how parent and child objects interact as well as how they are referenced. To help with this understanding, Figure 1.3 gives a graphical representation of the basic client-side JavaScript hierarchy.

As depicted in this diagram, all client-side objects are derived from either the `Window` or `navigator` objects. Considering that this is an object-based language, this structure makes complete sense. All objects on a given page are constructed within the browser's displaying window, hence all these objects are descendants of the `Window` object. By using the `Window` object, a programmer is allowed to access the various frames, documents, layers, and forms on a page, as well as many other objects and properties.

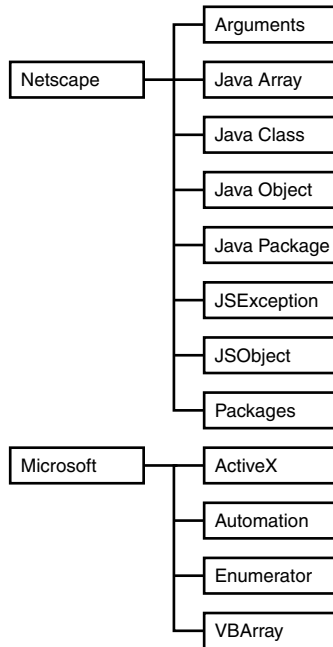
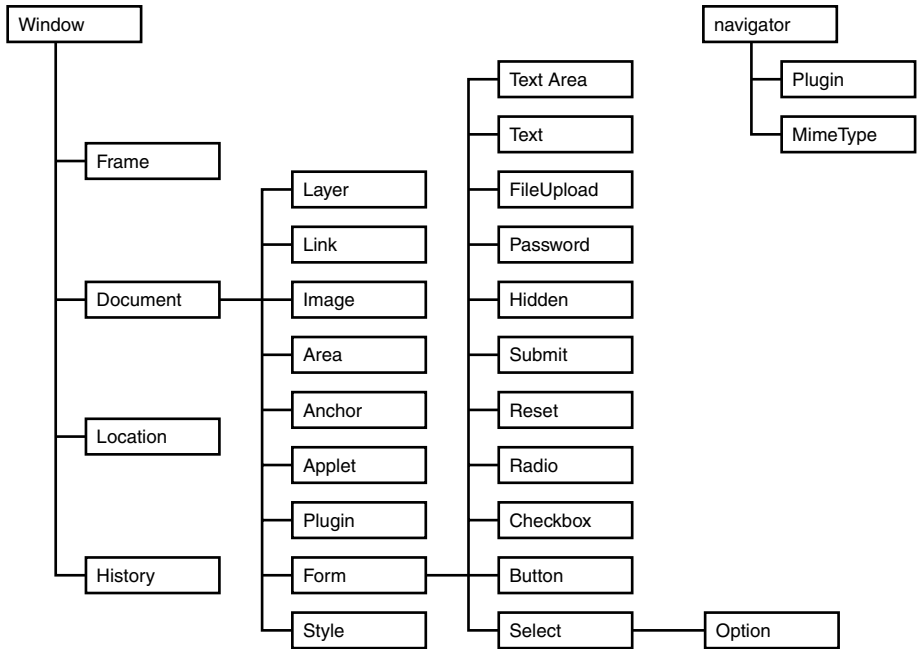


Figure 1.2

Core object hierarchy for Microsoft and Netscape environments.

The `navigator` object pertains to elements that are part of the browser itself. This specifically refers to the plug-ins installed and the *MIME* (*Multipart Internet Mail Extension*) types with which the browser is associated. Using the `navigator` object allows checking of the browser version, determining the plug-ins installed, and what programs are associated with the various MIME types registered on the system. There is also the ability to access other properties of the browser.

In addition to these client-side objects, browsers starting with Internet Explorer 5 and Netscape 6 have implemented features to support the *Document Object Model* (*DOM*). The DOM is a method in which documents can be referenced. We talk more about the DOM in Chapter 4, “Client-Side Scripting,” but wanted to include Figure 1.4 to show you the objects that make up this model.

**Figure 1.3**

Client-side JavaScript object hierarchy.

Server-Side

Similar to client-side, server-side JavaScript has several objects from which all other objects are derived. The root objects are the `DbPool` and database objects, for the Netscape and iPlanet implementations, from which you can create connections to a database, as well as access cursors, stored procedures, and the resultsets you generate. Within the Microsoft ASP environment, you have access to several other objects, such as `Response`, `Session`, and `Request`. Figure 1.5 shows the server-side object hierarchy.

NOTE

Programmers familiar with Java will find this very similar to the Java language. There are not as many objects/classes in the JavaScript language, but the structure and manner in which you access them are similar.

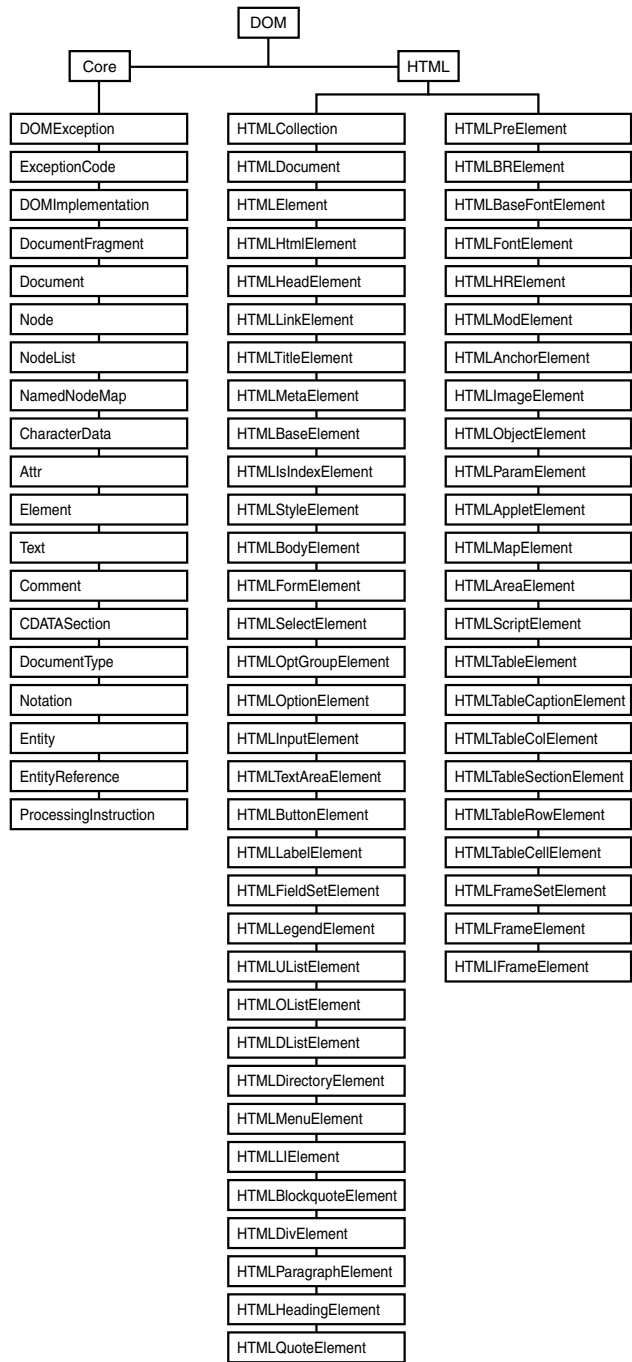
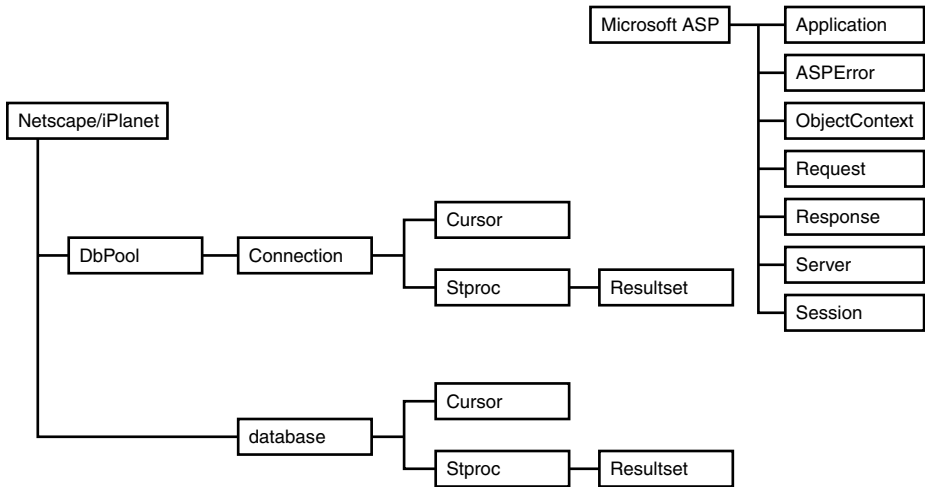


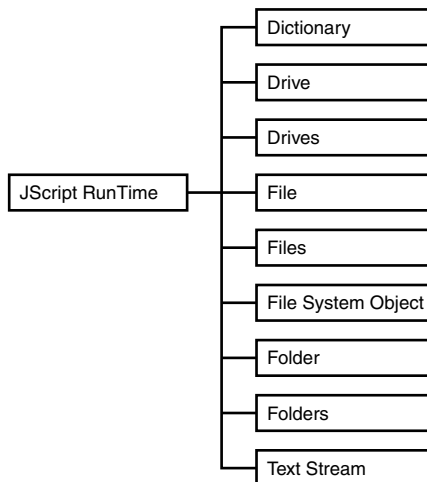
Figure 1.4
DOM object hierarchy.

**Figure 1.5**

Server-side object hierarchy.

JScript Runtime

As with about every other language that Microsoft has put its hands on, its JScript implementation contains objects that are specific to their runtime environment. Because their JScript engine is used within several of their applications and is a very important part of their .NET initiative, you will notice that many of the additional objects are familiar if you have ever worked with COM. Figure 1.6 illustrates the objects that are specific to the JScript implementation.

**Figure 1.6**

JScript RunTime object hierarchy.

Windows Script Host

In addition to the JScript-specific objects, Microsoft has also implemented objects that are part of its Windows Script Host environment. These objects, and associated code, are often used by administrators to perform everyday tasks previously done with batch files. The Windows Script Host, however, is much more powerful than the batch files of yesterday, and should be seriously considered for your administrative tasks. Figure 1.7 outlines the object hierarchy of the Windows Script Host objects.

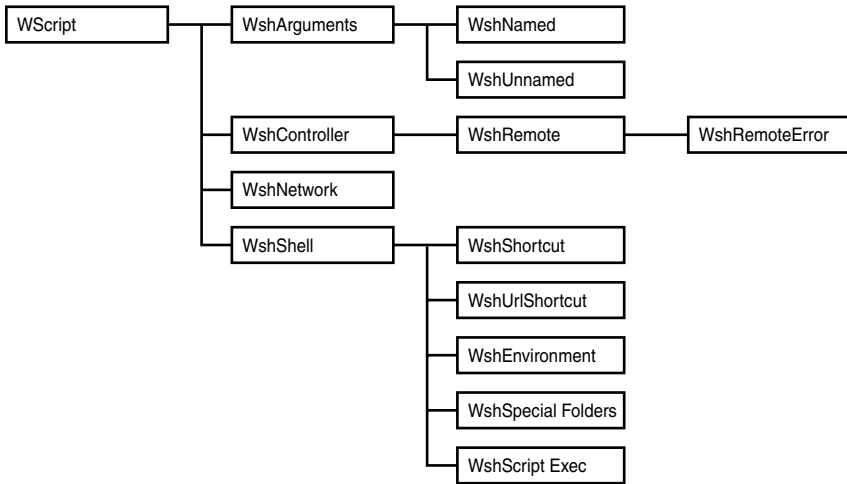


Figure 1.7

Windows Script Host object hierarchy.

Object Access

Because of this object hierarchy, accessing the various objects and elements on a page is accomplished by using the hierarchy itself. If you wanted to access a specific text field in a form on a page, you would do so using the following syntax:

```
window.document.formName.textboxName.value
```

TIP

With JavaScript, programmers have the ability to create their own objects or extend the core ones defined by the language. The explanation of how to create your own objects—and some examples—is covered in the section on functions in Chapter 2, “Details of the Language.” If you want to learn more about extending the functionality of the existing objects, look up objects with the *prototype* property in the reference section of this book.

Because JavaScript is object based, it automatically provides many advantages to using a modular approach to your programming. By creating your own objects and methods,

you are able to better maintain the code with which you are working. You will be creating code that can be reused in other programs, locations, and instances. Why write virtually the same code twice (or many times), when you can create it once and pass the characteristics that differentiate it from other, similar objects?

Modular Programming

To program in a modular fashion in JavaScript really involves three key things. Using these items in your programming will allow you to create code that can be reused from project to project. These are

- Creating your own objects
- Defining general functions to handle common tasks
- Placing reusable code in external JavaScript source files (commonly *.js files)

NOTE

As with any other language, remember that good comments within your code and documentation are often the most beneficial aspects of programming.

Because creating your own objects is discussed in Chapter 2, let's take a look at defining functions to handle common tasks. As with other programming languages, there are instances in which you have to perform certain processes over and over. Many times this might involve a different value of parameters passed, but the processes you go through are the same.

As an example, think of verifying a date entered by a user. This user is supposed to enter the month, date, and year in a form that will be submitted to your Web server for further processing. One of the concerns of the programmer is that he needs to have the date in a MM/DD/YYYY format, where the month and date need to be two characters and the year should be four.

To accomplish this task, you can create a single function that prepends a 0 in front of any single digit passed. This function could simply check to see if the value passed was less than the number 10, and, if so, it would perform the prepend. By defining this process in a function, a programmer will be able to use the same function for both the month and date verification. This avoids the trouble of writing a function for each. Even though this is a simple example, it illustrates the benefits of function and code reuse.

Programmers can also modularize their programming techniques by including their code in external JavaScript source files. This allows them to write code once, store it in a single location, and include it on many pages by simply referencing the location of the source file. If the function needs to change, they only have to change it in a single file and not every file that uses it. It is simple things such as these that save Web programmers hours or days of work time.

TIP

Do you serve ads on your Web site? If so, you might want to consider placing your ad tags in an external JavaScript source file. This will allow you to change the tags across your entire site if need be. There might be some browser issues with older versions, so if you make this decision, be sure you at least cover the browsers you wish to support.

Security

One of the biggest issues facing Internet development today is security. It is not possible to successfully develop any kind of application, whether it's Web based or based on the Web, and not have to implement some kind of security features. A program's security measures can ultimately determine how valuable the overall application is to a user. If the code can be tampered with or is subject to destruction from another program, the program will be subject to scrutiny and denial of use.

Because JavaScript is interpreted most often within a browser's environment, a user can be subject to malicious code. Browser's run off the operating system itself, meaning that it has access to a user's file system. This makes it feasible that a JavaScript program could take advantage of a hole in the browser's security measures to access the file system. After a programmer has accomplished this, many things are possible—even access to private documents or the ability to delete them altogether. This leaves a user at the mercy of a hacker.

Providing security for JavaScript scripts is actually twofold. One is that of responsibility, which lies with the programmer. A programmer must ensure that the script the user executes is not malicious. The second responsibility falls to the users themselves. Users should make the ultimate decision whether to run a script on their systems—this is something that must be implemented in the browser's functionality.

Because of these potentially destructive situations, there are various levels of security that users and programmers can rely on when programming in JavaScript. As discussed, some are the responsibility of the programmer, whereas others involve measures put in place by the browser that allow the user to control what is executed on his system.

What Security Measures Are in Place?

When JavaScript 1.0 was released in the Navigator 2.0, Internet Explorer 3.0 (JScript 1.0), and Opera 3.0 browsers, the only real security layer was that of a user having the ability to turn JavaScript on or off. The browser itself controlled the runtime environment for the language and any security measures it had in place.

In this model, when JavaScript was enabled, it was up to the browser to protect the user from any harmful code. Originally, this seemed like a thorough plan for implementing

security. *Leave it to the experts to protect the users.* However, where there is a will there is a way, and the first misuses of JavaScript began to surface.

One of the first items that seemed to be a potential problem occurred when frames were used on a Web site. Because frames load separate documents in each of the predefined areas, it is possible to load documents from several different domains and servers to make up the content displayed to the user. The problem arose when a document's JavaScript variables from one server were available for examination and modification on another. But this was only the start of the potential security holes that would follow.

To help protect users from the frame problem, Navigator 2, Internet Explorer 3, and Opera 3 implemented the *Same Origin Policy*. This policy prevented JavaScript code sent from one server from accessing properties of a document sent from another server, port, or protocol and returning that information to its original server.

Obviously, this policy does not affect all the elements of a given document, but it does include a core set. At the time of this printing, the document properties that must pass this origin check are in Table 1.1.

Table 1.1 Document Objects That Must Pass Origin Verification

Object	Property/Method
document	Read/Write: anchors, applets, cookie, domain, elements, embeds, forms, lastModified, length, links, referrer, title, URL, each form instance, each Java class available to JavaScript via LiveConnect Write: all other
image	lowsrc, src
layer	src
location	all except location.X and location.Y
window	find

Because it might be desirable for a script to access variables located on a page served from another server within the same domain, there is an exception to this security model. By definition, it would not be possible to access and upload document properties in a frame served from `http://myscripts.purejavascript.com` from another frame that was delivered from `http://mydocs.purejavascript.com`. Even though the domain is the same, the complete URL is not.

To get around this minor situation, programmers can set the `document.domain` property to the suffix of the current domain. This will allow them to access JavaScript properties on pages served from other servers within their domain. Following the example in the last paragraph, using the following line in the code can enable this feature:

```
document.domain = "purejavascript.com";
```

Setting this property will allow you to access the other *sub-domains* within your domain.

Data Tainting

When JavaScript 1.1 was released in Navigator 3.0, Netscape furthered its security implementation by using what is referred to as *data tainting*. In addition to the security model in the first generation JavaScript browsers, data tainting allowed the user and programmer to specify if they wanted scripts to access properties in other documents from other servers. When data tainting is not enabled, which is the default, the user will get a message saying that accessing document properties from other servers is not allowed.

NOTE

Data tainting was only implemented in JavaScript 1.1, and because the majority of users are now using browsers that support language versions higher than this, it is recommended not to use this security approach. However, this section is relative to understand some of the history of security within JavaScript, as well as answer some questions for anyone having to create code for Navigator 3 browsers.

Users can enable tainting if they want scripts on a page to have global access to other scripts and document properties. This is a security risk, but might be necessary within an enterprise environment in which other security measures are in place. To enable data tainting, environment variables must be set for the browser running the scripts. Table 1.2 shows how this can be accomplished on the various operating systems.

Table 1.2 How to Enable Data Tainting for Your Navigator Browser

Operating System	Environment Variable	Notes
Windows	NS_ENABLE_TAINT=1	Set this in the autoexec.bat for all Windows systems.
UNIX	NS_ENABLE_TAINT=1	Depending on which shell you are in, you will use some form of <i>set env</i> or <i>env</i> to set this variable.
Macintosh	Remove the two ASCII slash (//) comments before the NS_ENABLE_TAINT	This can be found by editing the resource with type Envi and number 128 in the Navigator application itself. It should be near the end.
OS/2	NS_ENABLE_TAINT=1	Set this in the config.sys.

After this variable is set, a number of document properties are affected. Table 1.3 shows a list of the document objects that are tainted by default.

Table 1.3 Document Objects That Are Tainted by Default

Object	Tainted Property
document	cookie, domain, forms, lastModified, links, referrer, title, URL
Form	action, name
each Form instance	checked, defaultChecked, defaultValue, name, selected, selectedIndex, text, toString, value
history	current, next, previous, toString
image	name
Link	hash, host, hostname, href, pathname, port, protocol, search, toString
location	hash, host, hostname, href, pathname, port, protocol, search, toString
Option	defaultSelected, selected, text, value
Plugin	name
window	defaultStatus, > name, status

TIP

As a programmer, you can test to see if the Navigator 3 user has tainting enabled by using the `navigator.taintEnabled()` method. See this entry in Chapter 8, “Client-Side” for an example of using this method.

In addition to the user having the ability to specify how he wants to handle tainting, a programmer can specify, or *taint*, objects or information that cannot be passed from one script to the next without the user’s permission. When this occurs, the browser will pop up a dialog box that allows the user to decide whether the information can be passed.

NOTE

For more information on data tainting, see the Client-Side JavaScript Guide on Netscape’s DevEdge (<http://developer.netscape.com>) site. There is an entire section (“Using Data Tainting in JavaScript 1.1”) in the “JavaScript Security” chapter devoted to security and the concepts of data tainting.

Because data tainting did not provide the true security model JavaScript needed, Netscape deprecated its functionality in JavaScript 1.2 and replaced it with Signed Scripts. This is the current and most complete model that has been implemented, and should be used by JavaScript developer’s moving forward.

Signed Scripts

Signed scripts allow a programmer the ability to gain access, after user authorization, to restricted information. This model, which was based on the signed objects model in

Java, uses LiveConnect and the Java Capabilities API to execute its functionality. Using this model gives programmers very defined control over what they can and cannot do on a user's machine.

TIP

More information on the Java Capabilities API can be found on Netscape's DevEdge site at <http://developer.netscape.com/docs/manuals/signedobj/capabilities>.

When using this model, you have the ability to sign external JavaScript source files (called through the `src` attribute of the `<script>` tag), event handlers, and code that is included inline on the page. The actual signing of these scripts is implemented by using Netscape's Page Signer tool, which is available at <http://developer.netscape.com>.

This Page Signer tool allows you to build a JAR (Java Archive) file that includes the programmer's security certificate and code. When the browser encounters a `<script>` tag that has an `archive` attribute set, it will go through the proper verification process before the script is executed. This process involves popping up a Java Security dialog box that gives the user the ability to grant or deny the rights to the script. The following is an example of syntax used on a page that includes a signed script:

```
<script src="myScripts.js" archive="sample.jar"></script>
```

If the code is inline, the JAR file will contain only the programmer's certificate. Calling the appropriate JAR file would then resemble the following, which does not have the `src` attribute, and it would have the code between the beginning and ending `<script>` tags.

```
<script archive="sample.jar" id="purejs">  
// Your code here  
</script>
```

Even though signed scripts are based on a Java model, there are enough differences in the languages that make it a bit harder to secure JavaScript code. Unlike JavaScript, a Java programmer can protect, make private, or make final variables and methods in their code. This inherently protects them from hackers because these elements cannot be accessed or changed—the Java language defines them as such.

Some expanded privileges can be accessed through the `netscape.security.PrivilegeManager.enablePrivilege()` method, which gives more control in scripting. This Java method allows a programmer to try to enable one of a set of privileges by asking the user to accept or reject his access. As with other signed scripts, this will prompt the user to grant or deny a programmer's request. The following list shows the privileges that a programmer can attempt to access for these purposes:

- `UniversalBrowserAccess`—Allows both reading and writing of privileged data in browser.

- **UniversalBrowserRead**—Allows the reading of privileged data in browser. This is required when using an `about:` (but not `about:blank`), getting any property of the `history` object, or getting the value of the `data` property of a `DragDrop` event within your scripts.
- **UniversalBrowserWrite**—Allows the writing of privileged data in browser. This is required when setting any property of an event object, adding or removing any of the browser's bars (location, menu, status, and so on), as well as using several of the methods and setting some of the properties of the `Window` object within your scripts.
- **UniversalFileRead**—Allows the script to read files on the file system of the machine on which it is running. This is required when using a file upload within your scripts.
- **UniversalPreferencesRead**—Allows the script to read browser preference settings.
- **UniversalPreferencesWrite**—Allows the script to write browser preference settings.
- **UniversalSendMail**—Allows the script to send mail under the user's name. This is required when using a `news:` or `mailto:` within your scripts.

JavaScript has quite an extensive list of security measures in place that can be used by the programmer. However, a programmer should use the security measures in a manner that maximizes his effectiveness. If this is not done, the scripts are subject to hacking.

Now that you have an understanding of the security measures in place for JavaScript, take a look at some of the overall advantages of using the language as a means of deploying solutions on the Internet or within an enterprise.

Advantages of JavaScript

Up to this point, you might not have seen any big reasons where and why JavaScript can help you. It is object based, can be interpreted within a browser, and there are security measures in place—but the same can be said for Java. Now that browsers support plug-ins and ActiveX controls, it is possible to design client-side functionality with more common languages such as C++ or Visual Basic. So what does JavaScript really give you?

For starters, it is platform independent. This is a major advantage over ActiveX controls and plug-ins because they have to be recompiled and potentially rewritten for the various platforms out there today. Previous versions of Navigator and the new Netscape 6, for example, run on many different platforms, and even though most of these are various flavors of Unix, at its core, you would still have to build a control or plug-in for Windows 16- and 32-bit systems, MacOS, Unix, BeOS, OS/2, and the list goes on. Also note that flavors of Unix and Linux can run on several types of processors (MIPS, Intel, PowerPC, and so on), and Windows NT runs on Intel and Alpha machines. This becomes quite an extensive list of components to maintain if you develop in a platform-dependant language.

Another advantage of JavaScript is that both Netscape and Microsoft Web servers have built-in interpreters. Both of these companies have implemented this in a different fashion, but, as a Web developer, you still have the ability to use the same language on the server-side that you do on the client-side. The only real competitor to JavaScript in this aspect is Java with its Java applet and servlet technology.

Platform Independence

Platform independence is probably the number one reason to use JavaScript within your applications. True, some environments interpret JavaScript a bit differently, but the majority of the language is processed the same. The code is interpreted so that you can write it once and let the execution environment interpret it.

This is a simple concept, but can be a big factor in deciding how to implement an application solution. As a programmer, you do not want to have to modify code to work on different operating systems or recompile for different microprocessors. You want to write the code once and be done with it. You want to be able to make changes easily and quickly without having to recompile 10 or 15 times. Let's face it; you want and need JavaScript.

Client-Side and Server-Side Versatility

The majority of the discussion so far has focused on using JavaScript on the client-side. Even with its initial release, JavaScript has also been implemented on the server-side within the Netscape/iPlanet and Microsoft Web servers. This server-side code contains many of the same objects and methods as the client-side, but it also has objects specific to the server environment—objects that allow you to connect to, query, and get results from a database. All this information is collected and processed before the server sends the page back to the requesting browser.

By providing this scripting capability on the server, a programmer can now use the language to dynamically build pages based on the execution of the server-side scripts it contains. Server-side JavaScript also can be used to maintain state for users as they move through a site. This maintaining of state is often implemented as a shopping cart on commercial sites. As users shop on a given site, server-side JavaScript can be used to track them and keep selected items in their carts.

Microsoft has also implemented a type of server-side JScript within its *Internet Information Server (IIS)*. Its implementation is used in *Active Server Pages (ASP)*, where the ASP filter parses a site's pages before they are sent back to the requesting browser. JScript is also a very important language within Microsoft's .NET initiative, where developers can use the language to create Web services that operate much the same as COM objects, but across the Internet. As with Netscape's implementation, this allows a Web developer to dynamically build the content of a page before it is sent back to the browser.

NOTE

Remember that JScript is Microsoft's equivalent to JavaScript.

Because of the functionality of these pages, ASP has given developers the ability to use JScript to call server-side components (such as ActiveX controls), pass the necessary parameters, and write the results to the screen. This allows a Web site to modularize all the functionality of building pages with individual components that are responsible for their specific tasks. JScript is used to handle the requests and results to and from these modules, and then write the results to the page.

When to Use JavaScript

One of the most important things to know about JavaScript is when to use it. Even though it provides much needed functionality in many scenarios, often it is simply not needed. One reason is the fact that JavaScript is not always interpreted the same or correctly—an important point to remember.

As a programmer, you should be able to write code, no matter how simple or complex, that will be executed correctly. However, there are browsers that have bugs that prevent JavaScript from working the way it was programmed. Before programming in JavaScript, you should first try to understand any documented bugs that exist. Doing so can save you hours of debugging in the long run. You can often find these bug lists on the Web, so check out our resource section toward the end of this chapter to get started.

Try to determine if you really need to use JavaScript on a given page as well. Ask yourself whether you are using it to add functionality to the page or just to make its appearance better. JavaScript can do a lot of neat things to a Web page, but, if it causes your page to break in certain browsers, you should avoid using it. A fine line exists between what you gain in functionality and what you expose as problems, so be sure to test your code with as many browsers and platforms as possible.

Depending on programmers' objectives when using JavaScript, they might be able to impose browser requirements. If they have stated that their pages only work in browsers later than Netscape 6 and Internet Explorer 5, it is safe for them to use JavaScript 1.3 or lower for their scripting needs. This immediately eliminates them from having to support older browsers, which can save many lines of code. Developers might not be able to impose these restrictions on a Web site, but it is likely that they can on Web-based applications.

Overall, programmers should be smart about using the language. They need to evaluate what their objectives are and who their audience is. When these requirements are defined, they can reverse engineer the project to determine what code they need to write. This is often a much easier approach than starting with an idea and trying to make it work in all circumstances.

Now that you've taken a quick look at some of the general issues to analyze before using JavaScript, take a look at what you can do with it. The following sections contain some of the common uses of the language, as as some more complex and specific uses.

Web Page Enhancements

Web page enhancements were the first real use of JavaScript. Any of you who have been working with the Internet since the release of Netscape Navigator 2 probably remember those annoying scrolling messages in the status bar of the browser window. This was one of the first enhancements done using JavaScript. Even though it became annoying, it definitely caught the eye of users.

Another popular item JavaScript is used for is writing the current date and time to a page. Some sites write the date and time the document was last modified, whereas others write the current date and time. This is widely used on sites that are news related in which the date of the document is very important to readers.

TIP

Writing the date and time to a page is a perfect item to modularize. If the code is written as a function, it can be included easily on all your pages and called when needed. If you are using some kind of browser intelligence on the server side, it is possible to include this function based on the browser that is requesting the page. If it can interpret JavaScript, make it part of the page. If it cannot, do not include it.

A final example of using JavaScript to enhance Web pages is to produce rollover buttons. This usually occurs on pages in which the linked images change when a user rolls over them. It is also possible to program in a down state when a user clicks and holds their mouse button down on the image. Even though this is a simple enhancement, it makes a page look and feel more professional. This effect allows a Web site to give the user the same experience as using his favorite application, be it a Web browser, a word processor, or a money manager.

These three implementations of JavaScript to enhance Web pages are pretty simple, but are by no means the limit of what can be done. Many sites have used JavaScript for advertisements, pop-up navigation windows, page redirects, and validating forms. Because the language is executed within the browser's environment and is often used to complement HTML publishing, there is virtually no limit to what can be done.

TIP

If you want to use JavaScript to enhance your Web pages, don't make the mistake of trying to think of something cool you can do with the language. You should try to reverse engineer it. Think of something cool for your site, and then figure out how to implement it in JavaScript.

Interactive E-Mail

Interactive e-mail is something that has come about within e-mail applications. It wasn't long ago that many of these programs were only able to read text e-mails. These programs now have the capability to render HTML e-mail within their interface, which

extends the formatting options a user can exploit. This not only improves the look and feel of the e-mail, but it also improves the readability of it. If a user wants something in italic, you can put it in italic.

Because HTML e-mail has become widely used in the Internet community, more and more e-mail applications are supporting it. In addition to HTML, Netscape and Microsoft's most recent e-mail applications support JavaScript within the body of an e-mail message (assuming that the user has it enabled). This makes it possible for a user to send HTML e-mails containing JavaScript that is interpreted when the recipient reads the message.

As a programmer, you need to keep in mind that an e-mail application is not a browser. Users are very particular about what they experience in their messages, and overuse of JavaScript could lead to annoying your recipients. JavaScript should be used sparingly in e-mails. It should be reserved for simple page enhancements such as image and link rollovers or calling ads within your message. Anything beyond this could cause problems when the application interprets your scripts.

Web-Based Applications

Web-based applications are probably the most useful instances of JavaScript. They allow a programmer to set user browser requirements, which in turn gives them a head start on the version of JavaScript they have at their disposal. This also results in limited exposure to browser bugs because programmers can define which browsers they support.

One of the most common uses of JavaScript within Web-based applications seems to be in controlling forms on a page. This can be anything from checking a user's values before submission to dynamically adjusting the values based on user-selected data. By implementing JavaScript at this level, a programmer is able to reduce the amount of user error when submitting forms. No more invalid credit card numbers because one digit too many was entered. No more usernames and passwords submitted as e-mail addresses, and no more incomplete forms.

JavaScript is also used in more full-blown Web-based applications. These applications are not necessarily for the common Internet user to experience, but rather are interfaces to enterprise level applications a company might have purchased. Some of the more common applications are used for reporting or ad delivery and management. Because the content on the application's pages is dynamic and always changing, a developer usually interfaces the application with a database or system process to build the pages on-the-fly. Using JavaScript allows developers to verify items before requests are made, as well as add an appealing look and feel to the application.

Windows Scripting

Microsoft's Windows Script Host comes with Windows 98, including Windows 2000, and can be installed in Windows 95 and NT 4 systems. This scripting host is language independent for ActiveX scripting on Windows 32-bit systems. *Language independent* means that a variety of programming languages can be used in conjunction with the

host. The reason it's mentioned in this book is that it natively supports JScript—Microsoft's implementation of ECMAScript.

NOTE

In addition to the JScript language, this scripting host also supports Visual Basic Script (VBScript) as well as other third-party languages such as Perl, REXX, TCL, and Python.

Using JScript in the scripting host allows an administrator or user to create scripts that perform various tasks on the operating system. These can be as simple as logon scripts or can be used to call ActiveX controls to perform more complex tasks. If you work in the Microsoft Windows environment, you will find this implementation of JScript can be very helpful.

TIP

For more information on the Windows Script Host, check out Microsoft's Developer Network site at <http://msdn.microsoft.com/scripting> and click the link to Windows Script Host.

JavaScript Resources

When you program a lot in a particular language, especially one that's Internet related, you come across many resources. Additionally, when you program a lot in a particular language, especially one that's Internet related, you need many resources. So to conclude this introductory chapter, we have included some resources for you. There's everything from general information to core documentation and references to news-groups—all on the JavaScript language and all online.

General Information

One of the most important types of resources in any given language is the general resource. Even if a book carries comprehensive coverage of a topic, it might not have conveyed the subject matter in a form that you understood. For this reason, you might want to study the same topic from a different person's perspective. Table 1.4 lists some resources that will allow you to do this.

Table 1.4 General Resources

Resource	URL
About.com Focus on JavaScript	http://javascript.about.com/compute/javascript/mbody.htm
DevEdge Online	http://developer.netscape.com/tech/javascript/index.html
Danny Goodman's JavaScript Pages	http://www.dannyg.com/javascript

Resource	URL
JavaScript.com	http://www.javascript.com
Developer.com	http://developer.earthweb.com/dlink.index-jhtml.72.1313.-.0.jhtml
Doc JavaScript	http://www.webreference.com/js
The JavaScript Workshop	http://www.starlingtech.com/books/javascript
JavaScript World	http://www.jsworld.com
Java/JavaScript Resources on the Internet	http://www.dezines.com/dezines/javainlinks.html
Microsoft Developer's Network	http://msdn.microsoft.com/scripting
Timothy's JavaScript Examples	http://www.essex1.com/people/timothy/js-index.htm
Using JavaScript and Graphics	http://www.javaworld.com/javaworld/jw-08-1996/jw-08-javascript.html
Using JavaScript's Built-In Objects	http://www.javaworld.com/javaworld/jw-05-1996/jw-05-javascript.html
Voodoo's Introduction to JavaScript	http://rummelplatz.uni-mannheim.de/~skoch/js/script.htm
WebCoder.COM	http://webcoder.com
Yahoo! Computers and Internet, Programming Languages, JavaScript	http://dir.yahoo.com/computers_and_internet/programming_languages/javascript
Open Directory Top, Computers, Programming, Languages, JavaScript	http://dmoz.org/Computers/Programming/Languages/JavaScript

Reference

Another important resource for any programmer is true reference documentation. This documentation represents information about that language as defined by standards or by companies who have built or implemented the language. Table 1.5 includes a list of online resources for the various reference documents out there today.

Table 1.5 Reference Resources

Resource	URL
Ecma-262 (ECMAScript)	http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM
Microsoft Scripting Technologies (JScript)	http://msdn.microsoft.com/scripting/default.htm?scripting/jscript/techinfo/jsdocs.htm

Resource	URL
Mozilla.org	http://www.mozilla.org/js
Netscape's Core JavaScript 1.4 Reference	http://developer.netscape.com/docs/ manuals/js/core/jsref/index.htm
Netscape's Server- Side JavaScript 1.2 Reference	http://developer.netscape.com/docs/ manuals/js/server/jsref/index.htm
Netscape's Client- Side JavaScript 1.3 Reference	http://developer.netscape.com/docs/ manuals/js/client/jsref/index.htm

Newsgroups

The final resource that we are going to discuss is the old standby—Usenet, or Newsgroups. Newsgroups are often a very good forum to post questions about problems and see responses to issues that others are having. It's global collaboration at its best and often is an overlooked resource. Table 1.6 lists some of our favorites, so be sure to check them out before giving up on any project.

Table 1.6 Newsgroup Resources

Server	Newsgroup
secnews.netscape.com	netscape.public.mozilla.jseng
secnews.netscape.com (Secure)	netscape.dev.js-debugger
	netscape.dev.jsref
	netscape.dev.livewire
	netscape.dev.livewire.dbconfig
	netscape.dev.livewire.programming
	netscape.dev.visual-javascript
	netscape.devs-livescript
Public Newsgroups	comp.lang.javascript

Moving On

This chapter covers the overview of the JavaScript language. As you can see, JavaScript is actually a very powerful scripting language that has many advantages. Security features are in place, and other implementations of the language make it worth any programmer's time to learn.

In the next chapter, you will take a look at the details of the language. Details that will give you, the programmer, an understanding of how the language deals with operators, data types, variables, functions, loops, conditionals, as well as how to correctly implement JavaScript within the body of an HTML document.



CHAPTER 2

Details of the Language

For experienced programmers to pick up a new language quickly, they look for similarities at the core of the new language and other languages they have used. These similarities generally include operators that enable programs to function, variables that provide memory, and the ability to apply the same operation to various items. Understanding how to use these core pieces of the language is essential if you want to begin programming in JavaScript.

If you have been programming for a long time, you might be tempted to skip over this chapter. Because JavaScript is still a young scripting language with some wrinkles to be ironed out, it is a good idea to understand these instances for backward compatibility reasons. Taking a little time to make sure that the core elements perform as you are expecting will save a lot of programming time in the future.

Things to Know about JavaScript Syntax

Before getting too deep into the core elements of the language, there are a few things a programmer should know about JavaScript syntax. Understanding these points will get you up and programming in a more timely fashion.

The Semicolon

If you have done any programming in C, C++, or Java, even as simple as a *Hello World* program, you already know 75% of all there is to know about the JavaScript semicolon (;). Just the same as C and C++, the semicolon is placed at the end of a JavaScript statement to signify that the code between the beginning of the line and the semicolon should be executed

before moving to the next portion of code. If you forget a semicolon at the end of a line in C++, you get compile errors, but JavaScript doesn't complain. Because JavaScript is a loosely typed language, forgetting a semicolon tells JavaScript to assume that you intended for one to appear at the end of the line, and it executes your code accordingly. This does not mean that you should not use semicolons! It is good programming practice to always include semicolons at the end of a line of code except when dealing with statements such as `for`, `while`, and `if`.

Although it is good programming practice to have only one functional piece of code per line, there are times when it is advantageous to put two independent pieces of code on one line. When this case arises, you must use a semicolon to separate the two pieces of code. In Listing 2.1, a semicolon is used to separate two independent pieces of variable declaration code that are placed on one line. Notice that semicolons were placed at the end of each line although JavaScript would do it for you. The result of executing the code is the phrase, "The sales tax on \$5 is \$.3", being displayed in the browser.

Listing 2.1 *Using Semicolons*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Declare 2 numeric variables on the same line
var fiveDollars = 5; var salesTax = 0.06;

// Compute the sales tax on 5 dollars and use the
// document.write() function to display the result.
document.write("The sales tax on $");
document.write(fiveDollars);
document.write(" is $");
document.write(fiveDollars*salesTax);

//-->
</script>
</html>
```

Using the <script> Tag

The first time a programmer works with a new language, he'll want to know the key pieces of syntax needed to start programming. In JavaScript, the HTML `<script>` tag is that key piece. `<script>` tags tell the browser that everything between `<script>` and `</script>` should be interpreted by the interpreter specified in the `type` attribute. There is no limit to the number of `<script>` tags that can be used, as long as they are used in pairs.

Notice that the browser interprets the code between the `<script>` tags based on the `type` attribute. Because the *type* attribute has no default value it is important that the

type attribute be set any time the `<script>` tag is used. To set the content type to the most current version of JavaScript supported by the browser, use the format `<script type="text/javascript">`. It is also possible to force the interpreter to use older versions of JavaScript (`<script type="text/javascript1.2">`, for example) as well as other languages such as Microsoft's JScript (`type="text/jscript"`).

NOTE

Before HTML 4, the *language* attribute of the `<script>` tag was the only way to set the scripting language. But as of HTML 4, the *language* attribute was deprecated in favor of the content *type* attribute discussed above. In order to ensure backward compatibility, both attributes are specified in the `<script>` tags throughout all the code examples in this book.

Comments

JavaScript is very generous with its commenting options by providing the `/* */` comment tags from C, the `//` comment tag from C++, and the `<!-- -->` tags from HTML. Just as in C and C++, the `/* */` enables comments to span multiple lines by just placing comments between the two tags. The `//` comment tag enables comments to be placed between the `//` and the end of the line. As mentioned earlier, JavaScript provides one other comment tag that might not be familiar to you, the HTML `<!--` comment. JavaScript interprets this comment the same way it interprets the `//` characters. You are probably asking yourself, "Why use two different comment tags that do the same thing?" Some older browsers that did not understand the `<script>` tags would display all the code between the `<script>` tags as standard HTML text. To prevent this with non-JavaScript-enabled browsers, `<!--` is placed on the line directly below the `<script>` tag, and `// -->` is placed on the line directly above the closing `</script>` tag. This causes non-JavaScript-enabled browsers to treat the code between the tags as HTML comments, but allows browsers with JavaScript interpreters to execute the code. Examples of this style of commenting can be seen in the examples throughout the book.

NOTE

The `//` comment characters have to be placed in front of the HTML `-->` comment closer because JavaScript will misinterpret `-->` as a pre-decrement operator.

Data Types and Variables

Before discussing JavaScript operators, conditionals, and loops, one should understand JavaScript data types and variables. These are building blocks that will be important going forward. Fortunately, JavaScript kept its implementation of data types simple and easy to use, unlike other programming languages. In addition to simple data types, variables are much easier to work with because there are no restrictions on the types of values they can hold.

Numbers

JavaScript’s approach to numbers is different from other languages because every number is treated as a floating-point number. JavaScript does support integers, octals, and hexadecimals from a formatting perspective, but, at the lowest level, JavaScript sees numbers as floating-point numbers. The following sections discuss different formats that numbers can have at the higher level.

Integers

Integers are numbers that contain no fractional parts, can be positive or negative, and can be formatted as a decimal, octal, or hexadecimal in JavaScript. Because integers are actually floating-point numbers in JavaScript, it is possible for the numbers to be very large.

- Decimal integers, also referred to as base 10, are probably the most common numerical values programmers use in their code. This type of integer is made up of numbers from 0 to 9 and cannot begin with leading zeros.
- Octal integers, also referred to as base 8, are a little different from decimal integers in that they must begin with a leading zero. Each digit following the leading zero can be 0 to 7.
- Hexadecimal integers, also referred to as base 16, must begin with 0x or 0X. Each digit following the leading zero can be 0 through 15, but 10 through 15 are represented by the letters a (or A) through f (or F).

Floating-Point Numbers

Unlike the integer, floating-point numbers can contain fractional parts and can use exponential notation for added precision. Floating-point numbers are made up of a decimal integer followed by a period (.) and the fractional portion of the number.

Exponential notation can be used by adding an e or E to the end of a floating-point number followed by a decimal integer that does not exceed three digits. This tells JavaScript to multiply the floating-point number by 10 to the exponent of the number following the e.

Built-in Values

Because computer programs are often used to solve scientific problems, the programs must know many of the numerical constants that are used in math and science. To make programming easier for you, JavaScript has included some of the more commonly used numerical constants in the `Math` object, which are shown in Table 2.1.

Table 2.1 Numerical Constants Provided by JavaScript

Math Constant	Description
<code>Math.E</code>	Base of natural logarithms
<code>Math.LN2</code>	Natural log of 2
<code>Math.LN10</code>	Natural log of 10
<code>Math.LOG2E</code>	Base 2 log of e

Math Constant	Description
<code>Math.LOG10E</code>	Base 10 log of e
<code>Math.PI</code>	Pi
<code>Math.SQRT1_2</code>	Square root of 1/2
<code>Math.SQRT2</code>	Square root of 2

Special Values

JavaScript also provides some special values that are common in the mathematical world but not so common in the computer world. These special values are available through the `Number` object, as shown in Table 2.2.

Table 2.2 *Special Numerical Values*

Number Constant	Description
<code>Number.MAX_VALUE</code>	Largest representable number
<code>Number.MIN_VALUE</code>	Smallest representable number
<code>Number.NaN</code>	Not a number
<code>Number.POSITIVE_INFINITY</code>	Positive infinity
<code>Number.NEGATIVE_INFINITY</code>	Negative infinity

Strings

Strings provide programs a voice with which to communicate. It would be inconceivable to create a programming language today that did not use strings because they are so important.

JavaScript Strings

In the world of C and C++, dealing with strings is like having to go to the dentist—dreaded! Dealing with strings in JavaScript is like going to a big candy store. A string is made up of any number of characters or a lack of characters. Strings are declared by placing the characters that make up the string between a pair of double quotes (" ") or single quotes (' '). What if a string contains double quotes or single quotes? No problem. JavaScript interprets single quotes as part of the string if the single quotes are inside a pair of double quotes. Likewise, double quotes are considered part of the string if they appear between a pair of single quotes. If single quotes are your only option for declaring a string that contains single quotes, or if double quotes must be used to declare a string that contains double quotes, you will need to use escape sequences (see the next section, “Special Characters”).

Special Characters

Just as in C and C++, escape sequences, which are noted by a backslash character (\), allow special characters that cannot normally be stored in a string to be declared. Table 2.3 lists all the possible escape characters.

Table 2.3 *Escape Sequences and Their Associated Characters*

Escape Sequence	Character
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\\</code>	Backslash
<code>\xxx</code>	Character represented by three octal digits xxx (000 to 377)
<code>\xx</code>	Character represented by two hexadecimal digits xx (00 to FF)
<code>\uXXXX</code>	Unicode character represented by four hexadecimal digits XX (0000 to FFFF).

Other Data Types

Outside of the world of computers, there are uncertainties and indefinable values that we come in contact with daily. When computer programs are written to simulate the world we live in, they must handle uncertainties and values that have no definition. JavaScript provides some special data types to handle these situations.

Boolean

The Boolean data type is much simpler than any of the other data types because it has only two possible values: `true` and `false`. Sometimes it is easier to think of `true` as on or yes and `false` as off or no when working with some expressions that use the Boolean data type. In JavaScript, `true` and `false` are often represented by `1` (`true`) and `0` (`false`).

null

JavaScript provides the keyword `null` for representing a condition in which there is no value. In some languages, `null` and `0` are considered the same value, but JavaScript sees `null` and `0` as two completely different values.

Undefined Values

At this point, you might be thinking that `undefined` and `null` are essentially the same, but this is not true. In fact, `undefined` is a concept rather than a keyword like the `null` data type. Undefined is equivalent to `NaN` for numbers, the string `undefined` for strings, and `false` when dealing with Boolean values.

What to Know About Variables

Computer programs would not do much if they did not have some type of temporary memory. Variables provide a way for data to be stored during the execution of a

program. Some languages, such as C and C++, impose many restrictions on how variables are used, but JavaScript keeps variables simple and easy to use.

Naming Variables

One of the keys to writing great code is to use variable names that help you, and programmers that modify your code, remember what data is stored in the variable. Before beginning to think of great variable names, remember the following guidelines imposed on variable names by JavaScript:

- The first character of the name must be a letter or an underscore (_).
- All characters following the first character can be letters, underscore, or digits.
- Letters can be either upper- or lowercase. JavaScript does distinguish between the two cases. For example, a variable called `jobTitle` is different from a variable called `JOBtitle`.

Assigning Values

When the perfect variable name has been derived, it is time to declare that variable and assign it a value. To declare a variable, use the keyword `var` followed by the variable name. Some programmers prefer to keep their code compact by declaring multiple variables using the same `var` statement. When this is the case, the variable names are separated by commas. At this point, the variable is undefined because no value has been assigned to it. Keep in mind that `undefined` is a special JavaScript value.

Now that the variable is declared, a value can be assigned to it using the assignment operator (`=`). In many cases, the declaration and assignment steps are performed in one step. If a value is assigned to a variable that has not been declared using the `var` keyword, JavaScript will automatically create a global variable. Listing 2.2 demonstrates the ways variable declaration and assignment can be performed. The code displays the sentence "James is 49 and 6 feet tall." in the browser window.

Listing 2.2 Variable Declaration and Assignment

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Variable declaration without assignment
var firstName;

//Variable assignment without declaration
firstName = "James";

//Variable declaration and assignment
var age = 49, height = 6;

//Display the results
document.write(firstName, " is ",age," and ",height," feet tall.");
```

Listing 2.2 Continued

```
//-->
</script>
</html>
```

TIP

Always use the `var` keyword to declare all variables to prevent variable scope problems.

Scope

A variable can be either global or local in JavaScript. All variables are global unless they are declared in a function; in which case the variable is local to that function. It is possible for two variables with the same name to exist if one is global and the other is local to a function. When accessing the variable from within the function, you are accessing the local variable. If the variable is accessed outside the function, the global variable is used (see Listing 2.3).

CAUTION

Always use the `var` keyword to declare local variables in functions. Without `var`, JavaScript will create a global variable.

Listing 2.3 Variable Scope Example

```
<html>
<h2><u>Computer monitor specifications</u></h2>

<script type="text/javascript" language="JavaScript">
<!--

//Initialize global variables
color = "green";
var size = 15;

//Declare a monitor specification function
function monitorSpecs()
{
    //Declare and set variables inside function
    color = "purple";
    price = "$300.00";
    var size = 17;
    document.write("The ",size," inch ",color);
    document.write(" monitor is ",price);
}

//Display results of monitorSpecs() function
monitorSpecs();
```

```
//Display variable values outside of function
document.write("<br>The ",size," inch ",color);
document.write(" monitor is ",price);

//-->
</script>
</html>
```

The results of running this script within the body of an HTML document can be seen in Figure 2.1. The code begins by declaring two global variables, *color* and *size*. The *monitorSpecs()* function creates a new variable called *size* that only exists within the scope of the function. Because the function did not specify *var*, the global variable *color* was changed from green to purple. In addition, a new global variable, *price*, was declared within the function because the word *var* was not used.

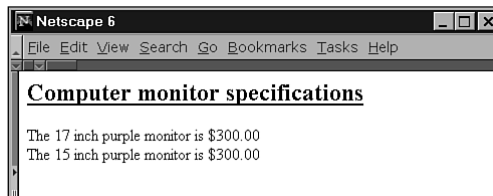


Figure 2.1

The 17-inch monitor has the same color price due to the use of global variables.

Type Conversion

In languages such as C and C++, type conversion is very important and complicated, but, in JavaScript, type conversion is effortless. Unlike other languages, JavaScript allows a variable to hold any data type at any time. This means that a variable can be assigned a string initially, but then could later contain an integer. JavaScript also attempts to perform all necessary type conversions for you, such as strings to numbers and numbers to strings.

Arrays

Arrays enable programmers to store multiple data, based on a numbered position called an index, into one storage structure. The numbering of the index always starts at 0 and goes up. Also, JavaScript supports having arrays within arrays, called *multi-dimensional* arrays.

The implementation of arrays in JavaScript has been changing ever since JavaScript was introduced. The original implementation of arrays in JavaScript 1.0 was not really an array at all but rather JavaScript objects with multiple property settings. A true Array object was added in JavaScript 1.1, and additional features added in following versions. Today, arrays are very robust and full featured, but, because of their changing past, you should spend some time digging into the history of arrays as they apply to JavaScript versions. You will begin by understanding how arrays work in the latest

versions of JavaScript and then go back and learn how arrays were created in JavaScript 1.1

One-Dimensional

To create an instance of an array, you must use the `new` operator along with the `Array` object. There are four ways to declare an array. First, an empty array that contains no elements can be created by leaving the constructor parameters empty as shown in the following JavaScript statement:

```
var x = new Array();
```

The second way to create an array is to fill in the constructor parameters with the array elements. One of the nice things about JavaScript arrays is that an array can contain elements of various types. For example, the following JavaScript statement creates an array that contains three strings, "red", "yellow", and "green", as well as the integers 1, 5, and 8:

```
var x = new Array("red", "yellow", "green", 1, 5, 8);
```

The third way to create an array is to fill in the constructor parameter with just the size of the array. This causes the array to be initialized to hold the number of elements specified, but does not specify the actual elements. For example, the following JavaScript statement creates an array that can hold 6 elements.

```
var x = new Array(6);
```

NOTE

The `var x = new Array(n);` format, described previously, is not recognized by JavaScript 1.2, so the number specified in the constructor parameter is stored as an element in position 0.

The fourth, and quickest, way to create an array is to use the standard array square brackets to fill in the array elements directly:

```
var x = ["red", "yellow", "green", 1, 5, 8];
```

After an array has been created, it can be written to and read from using the `[]` operator. By placing a position number in this operator, the data stored at this index can be accessed and even overwritten.

String Indexes

So far, you have only accessed elements in arrays via the numerical index, but it is possible to index arrays using strings. To access an element, a string index value is placed into the `[]` operator. Listing 2.4 demonstrates the use of strings as indexes for a clothing store's product quantity array. Figure 2.2 displays the clothing store's current inventory of products.

Listing 2.4 Using Strings for Array Indexes

```

<html>
<h2><u>Clothing Store Inventory</u></h2>

<script type="text/javascript" language="JavaScript">
<!--

//Populate an array with product quantities
function populateArray(products)
{
    products["shirts"]=46;
    products["pants"]=23;
    products["hats"]=14;
    products["socks"]=153;
}

//Display product quantities
function displayArray(products)
{
    document.write(products['shirts'], " shirts.<br>");
    document.write(products['pants'], " pants.<br>");
    document.write(products['hats'], " hats.<br>");
    document.write(products['socks'], " pairs of socks.");
}

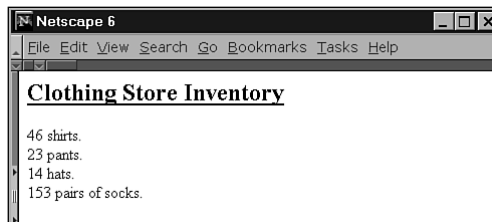
//Create a product quantity array
var productQty = new Array();

//Set product quantities
populateArray(productQty);

//Display the product quantities
displayArray(productQty);

//-->
</script>
</html>

```

**Figure 2.2**

The clothing store's current inventory of products is displayed using string indexes to access arrays.

Length

Unlike arrays in C and C++, JavaScript allows the size of an array to change dynamically at any time. For example, it is possible to write directly to a position that was not even declared using the [] operator. The length of the array can also be changed by altering the length attribute of the Array object. If the length of an array is originally 10 and is reduced to 5 by changing the value stored in the length attribute, the elements in position 5 through 9 are lost.

One of the advantages to using the Array object is the methods it provides to manipulate and access itself. Table 2.4 lists some of the methods that are currently available in the Array object. Details of these methods can be found in Chapter 7, “Core Language,” in the “Array” section.

Table 2.4 Methods Available in the Array Object

Method	Description
join()	Concatenates all elements into one string
reverse()	Reverses the order of the elements in the array
sort()	Sorts elements in array
concat()	Concatenates an array on to an array
slice()	Returns a subsection of the array
splice()	Inserts and removes elements from an array
push()	Adds elements to the end of an array
pop()	Deletes the last element from an array
unshift()	Adds elements to the front of an array
shift()	Deletes elements from the front of an array
toString()	Converts elements to a string

Multidimensional

To create multidimensional arrays in JavaScript, the element of an array must be another array. The inner array can be accessed by putting two [] operators back to back. Listing 2.5 uses a multidimensional array to hold an inventory list of brake parts. As seen in Figure 2.3, the brake parts list is accessed by using double [] operators and displayed in a table.

Listing 2.5 Using a Multidimensional Array

```
<html>
<h2><u>Brake Parts Inventory List</u></h2>

<script type="text/javascript" language="JavaScript">
<!--

//Display brake part inventory in a table
function displayInventory(table)
{
    document.write("<table border=on>");
```



```

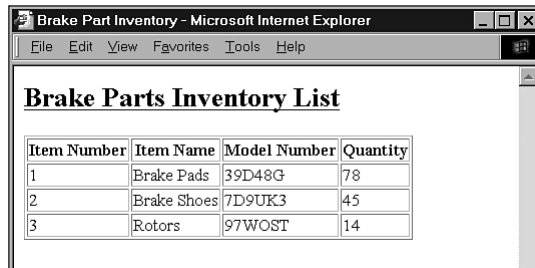
document.write("<th>Item Number</th><th>Item Name</th>");
document.write("<th>Model Number</th><th>Quantity</th>");
//Display each part
for(x=1; x<=3; x++)
{
    document.write("<tr><td>",x,"</td>");
    //Display all information for each part
    for(y=0; y<=2; y++)
    {
        document.write("<td>",table[x][y],"</td>");
    }
    document.write("</tr>");
}
document.write("</table>");
}

//Create a brake parts inventory list using a
//multidimensional array
part1 = new Array("Brake Pads", "39D48G", 78);
part2 = new Array("Brake Shoes", "7D9UK3", 45);
part3 = new Array("Rotors", "97WOST", "14");
brakeParts = new Array("", part1, part2, part3);

//Display the inventory of brake parts
displayInventory(brakeParts);

//-->
</script>
</html>

```



Item Number	Item Name	Model Number	Quantity
1	Brake Pads	39D48G	78
2	Brake Shoes	7D9UK3	45
3	Rotors	97WOST	14

Figure 2.3

Using a multidimensional array to display the inventory list of brake parts.

Arrays as Objects

Because arrays are essentially JavaScript objects, it is possible to access the elements of arrays as properties if a string index is used. Dot notation is used, rather than the [] operators. For example, the clothing store example, in Listing 2.4, could have been

created using properties as shown in Listing 2.6. Notice how the `[]` operators and dot notation are used interchangeably when accessing the contents of the array. The result of executing the code is the same as previously shown in Figure 2.2.

Listing 2.6 Accessing Array Properties

```
<html>
<h2><u>Clothing Store Inventory</u></h2>

<script type="text/javascript" language="JavaScript">
<!--

//Populate an array with product quantities
function populateArray(products)
{
    products.shirts=46;
    products.pants=23;
    products["hats"]=14;
    products["socks"]=153;
}

//Display product quantities
function displayArray(products)
{
    document.write(products['shirts'], " shirts.<br>");
    document.write(products['pants'], " pants.<br>");
    document.write(products.hats, " hats.<br>");
    document.write(products.socks, " pairs of socks.");
}

//Create a product quantity array
var productQty = new Array();

//Set product quantities
populateArray(productQty);

//Display the product quantities
displayArray(productQty);

-->
</script>
</html>
```

JavaScript 1.0 Arrays

As mentioned earlier, JavaScript originally used the `Object()` constructor to create arrays in JavaScript 1.0. Because the properties of an `Object()` could be accessed by using the `[]` operator, it was possible to give the illusion of an array.

To create an array using this concept, a new object is created using the `Object()` constructor. Once created, elements can be assigned to the object using the `[]` operators.

Because this is just a basic object, the programmer is responsible for keeping track of the length of the array. The easiest way to remember the length is to create a property called `length`. Unfortunately, properties use the same positions that are accessed by the `[]` operator, so the `length` property would actually be array position `0`.

Listing 2.7 demonstrates how to create an array representing a toolbox using the `Object()` constructor. The code displays the sentence "The toolbox holds: hammer wrench nails" in the browser window.

Listing 2.7 *Creating Arrays in JavaScript 1.0*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Create a toolbox array using the Object() constructor
var toolbox = Object();
toolbox.length=3;    //array position zero
toolbox[1]="hammer";
toolbox[2]="wrench";
toolbox[3]="nails";

//Display the items in the toolbox.
document.write("The toolbox holds: ");
for(x=1; x<=toolbox.length; x++)
{
    document.write(toolbox[x], " ");
}

//-->
</script>
</html>
```

Operators

JavaScript provides most of the common operators that can be found in other programming languages. Because of JavaScript's way of handling strings, some of these operators are a bit easier to use than in other languages.

Arithmetic

Just the same as other programming languages, JavaScript allows many arithmetic operations. These operations include the common addition and subtraction that all programmers use as well as the less common modulus and incremental.

NOTE

All the common arithmetic operators will attempt to convert strings to numbers when applicable. If a string cannot be converted to a number, `NaN` (**Not A Number**) will be returned.

Those who have programmed in other languages will find that JavaScript is very robust in its support of operators and mathematical functions. This is not only because of the built-in operators, but also because of the access to advanced mathematical operations that are provided through the `Math` object. The functions of this object are shown in Table 2.5 and are covered in Chapter 7.

Table 2.5 Advanced Mathematical Methods

Method	Description
<code>Math.abs()</code>	Absolute value
<code>Math.acos()</code>	Arc cosine
<code>Math.asin()</code>	Arc sine
<code>Math.atan()</code>	Arc tangent
<code>Math.atan2()</code>	Arc tangent
<code>Math.ceil()</code>	The smallest integer that is greater than or equal to a number (ceiling).
<code>Math.cos()</code>	Cosine
<code>Math.exp()</code>	Natural exponent
<code>Math.floor()</code>	The largest integer that is equal or less than a number (floor).
<code>Math.log()</code>	Natural logarithm
<code>Math.max()</code>	The larger of two numbers
<code>Math.min()</code>	The smaller of two numbers
<code>Math.pow()</code>	Power of
<code>Math.random()</code>	Random number
<code>Math.round()</code>	Round
<code>Math.sin()</code>	Sine
<code>Math.sqrt()</code>	Square root
<code>Math.tan()</code>	Tangent

Addition

The addition operator (+) is, of course, one of the most widely used and common operators. If the values on either side are numerical values, the values are added together. When the values are strings, they are concatenated together. The following line of code

```
var resultOfAdd = 34 + 12;
```

would set the variable `resultOfAdd` equal to 46, whereas this line of code

```
var resultOfAdd = "a" + "corn";
```

would set the variable `resultOfAdd` equal to the string "acorn".

Subtraction

The subtraction operator (-) subtracts the number to the right of the operator from the number on the left. When either of the operands are strings, an attempt is made to convert the strings to numbers. For example, the following lines of code

```
var aNum = String("102");  
var resultOfSub = 25 - aNum;
```

convert the string stored in *aNum* to a number before performing the subtraction operation. The result of the subtraction (-77) is then stored in the variable *resultOfSub*.

Multiplication

The multiplication operator (*) works the same as it would in any other language by multiplying the left operand by the right operand. The multiplication operator is no different from subtraction in its efforts to handle strings. If either of the values is a string, an attempt is made to convert the string to a number. For example, the following lines of code

```
var aNum = String("7");  
var resultOfMult = 5 * aNum;
```

convert the string stored in *aNum* to a number before performing the multiplication operation. The result of the multiplication (35) is then stored in the variable *resultOfMult*.

Division

The division operator (/) is the operator that, although simple, can be confusing when you have been writing code all day and your senses are dulled. You ask yourself, “Which number divides into the other?” Reading the expression from left to right, the left value is divided by the right value. As before, if either of the operands is a string, an attempt is made to convert the string to a number. For example, the following lines of code

```
var aNum = String("7");  
var resultOfDiv = 42 / aNum;
```

convert the string stored in *aNum* to a number before performing the division operation. The result of the division (6) is then stored in the variable *resultOfDiv*.

Modulus

Although the modulus operator (%) is not used as often as some of the other operators, I am always excited when I do get to use it because it usually means I am performing a neat math trick. This operator starts similar to the division operator, by dividing the left value by the right, but, instead of returning the normal result of division, only the remainder is returned by the operation. Once again, if either value is a string, an attempt is made to convert the string to a number. For example, the following lines of code

```
var aNum = String("3");  
var resultOfMod = 26 % 3;
```

convert the string stored in *aNum* to a number before performing the modulus operation. The remainder of 2 is then stored in the variable *resultOfMod*.

Pre-Increment

The pre-increment operator (`++`) combines two very common steps that programmers use over and over again into one, thus making code more concise and readable. This operator is especially handy when working with `for` loops. In your code, the pre-increment operator is placed directly before the variable to be incremented. The operation begins by incrementing the variable by 1. The new incremented value is returned by the operation to be used in another expression. If the variable is a string, it is converted to a number. For example, the following segment of code

```
//The price is $5.00 dollars  
var price = String("5");
```

```
//Add the shipping rate ($3.00 dollars) to the price after incrementing price  
var pricePlusShipping = (++price) + 3;
```

converts the string stored in *price* to a number before performing the pre-increment operation. The pre-increment operation results in the variable *price* being changed from 5 to 6 dollars, and the value of 9 dollars is stored in the variable *pricePlusShipping*.

Post-Increment

The post-increment operator (`++`) has the same operator as the pre-increment operator but it behaves differently based on its position. First, the post-increment operator appears directly after the variable that is to be incremented. Unlike the pre-increment operator, the post-increment operator returns the original value before it is incremented by 1. If either of the values is a string, an attempt is made to convert the string to a number. For example, the following segment of code

```
//The price is $5.00 dollars  
var price = String("5");
```

```
//Add the shipping rate ($3.00 dollars) to the price before incrementing price  
var pricePlusShipping = (price++) + 3;
```

converts the string stored in *price* to a number before performing the post-increment operation. The post-increment operation causes the variable *price* to be changed from 5 to 6 dollars but the original price of 5 dollars is added to the shipping rate resulting in the value of 8 dollars being stored in the variable *pricePlusShipping*.

Pre-Decrement

The pre-decrement operator (`--`) is very similar to the pre-increment operator in its placement to the left of a variable and its order of execution. But there is one key difference between the operators: the pre-decrement operator decrements the value by 1. Once again, if the variable is a string, it is converted to a number. For example, the following segment of code

```
//The price is $20.00 dollars  
var price = String("20");
```

```
//Subtracted discount ($6.00 dollars) from the price after decrementing price
var priceMinusDiscount = (--price) - 6;
```

converts the string stored in *price* to a number before performing the pre-decrement operation. The pre-decrement operation would result in the variable *price* being changed from 20 to 19 dollars, and the value of 13 dollars being stored in the variable *priceMinusDiscount*.

Post-Decrement

The post-decrement operator (`--`) is very similar to the post-increment operator in its placement to the right of a variable and its order of execution. But, as the name implies, the post-decrement operator decrements the value by 1. If the variable is a string, it is converted to a number. The following segment of code

```
//The price is $20.00 dollars
var price = 20
```

```
//Subtract the discount ($6.00 dollars) from the price before decrementing
price
var priceMinusDiscount = (price--) - 6;
```

converts the string stored in *price* to a number before performing the post-decrement operation. The post-decrement operation causes the variable *price* to be changed from 20 to 19 dollars but the original price of 20 dollars is used to calculate the value of 14 dollars that is stored in the variable *priceMinusDiscount*.

Unary Negation

The unary negation operator (`-`) is usually used when performing a mathematical equation in which a number needs to be changed from positive to negative or vice versa. When negating a variable, keep in mind that the contents of the variable do not change, only the value returned is negated. As with all the other operators, if the value is a string, an attempt is made to convert the string to a number. For example, the following segment of code

```
var aNumber = String("67");
var resultOfNeg = -aNumber;
```

converts the string stored in *aNumber* to a number before performing the negation operation. The result of negation on the number results in the value of -67 being stored in the variable *resultOfNeg*.

String

The addition operator (`+`) has a special purpose when dealing with strings. If the values on either side of the addition operator are strings, the strings are concatenated together. If only one of the values is a string, the other value is converted to a string and concatenated with the first value. To help understand these various combinations of applying the addition operator to numeric and string values, see Listing 2.8.

Listing 2.8 Using the Addition Operator on Numeric and String Values

```

<html>
<script type="text/javascript" language="JavaScript">
<!--

// Declare 2 numeric variables and 2 string variables
var sStringVar1 = "Hello";
var sStringVar2 = "World";
var nNumVar1 = 5;
var nNumVar2 = 10;

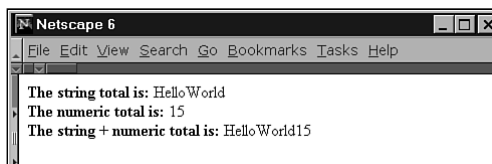
// Apply the addition operator to create 3 totals
var sStringTotal = sStringVar1 + sStringVar2;
var nNumTotal = nNumVar1 + nNumVar2;
var sStringNumTotal = sStringTotal + nNumTotal;

// Use the document.write() function to write the totals to the page
// Notice that we even use the addition operator in place of a comma
// to concatenate the results with some text on the page.
document.write("<b>The string total is: </b>" + sStringTotal + "<br>");
document.write("<b>The numeric total is: </b>", nNumTotal, "<br>");
document.write("<b>The string + numeric total is: </b>", sStringNumTotal);

//-->
</script>
</html>

```

The results of running this script within the body of an HTML document can be seen in Figure 2.4. As the figure shows, when the addition operator is applied to two strings or a string and a numeric value, a string concatenation occurs. As expected, when applying this operator to the two numeric values, the numbers are added.

**Figure 2.4**

Using the addition operator to add numbers and concatenate strings.

Assignment

What good are variables if data cannot be assigned to them? Similar to all languages, JavaScript provides assignment operators to allow data to be stored in variables. The basic format of the assignment operator is shown in the following example, where a value of 6 is assigned to the variable `x`:

```
x = 6;
```


In addition to this one-to-one assignment, this operator can also be stacked to create simultaneous assignments. Simultaneous means that several variables can be assigned at once. This is demonstrated in the following example, where variables *x*, *y*, and *z* all contain the value of 6:

```
x = y = z = 6;
```

Anytime multiple assignment operators occur in the same expression, they are evaluated from right to left. So in the previous example, *z* would be assigned the value 6 first. After the assignment has been made, *y* would be assigned the value stored in *z* and, finally, *x* would be assigned the value stored in *y*. The overall effect is that all three variables would be assigned a value of 6.

Because the assignment operator is an operator, similar to addition (+) or subtraction (-), it can be used within an expression. This enables a programmer to perform an assignment and evaluate a mathematical expression all in one step.

```
y = (x = 3) + 4;
```

In the preceding example, the value 3 is assigned to the variable *x*, which is then added to the value 4 and assigned to the variable *y*. After the expression is fully evaluated, *y* will contain the value 7. This enables programmers to accomplish three operations at once. They are able to assign a value to the *x* variable, perform addition, and assign the result to the *y* variable. Features such as this help make JavaScript very versatile and easy to use.

Now that you have looked at how the assignment operator works, look at a more detailed example—one that performs each of the assignments discussed, as well as some more complex ones. Listing 2.9 contains such an example, and Figure 2.5 shows the result.

Listing 2.9 Use of the Assignment Operators in JavaScript

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Declare variables using single assignment
x = 3;
y = 7;
z = 9;

//Display the values stored in the variables after single assignment
document.write("<u>After single assignment</u><br>");
document.write("x=",x,"<br>y=",y,"<br>z=",z,"<br>");

//Perform multiple assignment on variables
x = y = z = 14;
```

Listing 2.9 Continued

```
//Display the values stored in the variables after multiple assignment
document.write("<u>After multiple assignment</u><br>");
document.write("x=",x,"<br>y=",y,"<br>z=",z,"<br>");

//Perform multiple assignment in one expression
x = (y = 17) + (2 * (z = 2));

//Display the values stored in the variables after multiple assignment
//in one expression.
document.write("<u>After multiple assignment in one expression</u><br>");
document.write("x=",x,"<br>y=",y,"<br>z=",z,"<br>");

//-->
</script>
</html>
```

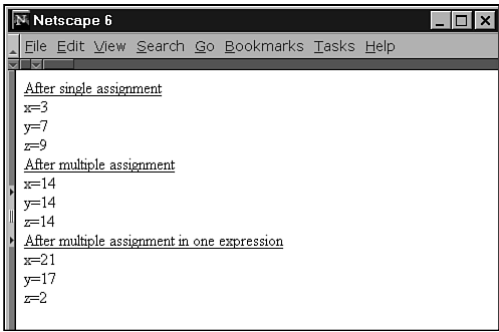


Figure 2.5
The result of various assignment operations.

In addition to the basic assignment operator, JavaScript also offers a number of advanced assignment operators that extend assignment functionality. These operators combine the functionality of basic assignment and other operators into one functional operator. Table 2.6 shows these advanced assignment operators along with their equivalent operations. Details of these assignment operators can be found in Chapter 7.

Table 2.6 *Advanced Assignment Operators*

Operator	Example	Description
+=	x+=y	x = x + y;
-=	x-=y	x = x - y;
=	x=y	x = x * y;
/=	x/=y	x = x / y;

Operator	Example	Description
<code>%=</code>	<code>x%=y</code>	<code>x = x % y;</code>
<code><<=</code>	<code>x<<=y</code>	<code>x = x << y;</code>
<code>>>=</code>	<code>x>>=y</code>	<code>x = x >> y;</code>
<code>>>>=</code>	<code>x>>>=y</code>	<code>x = x >>> y;</code>
<code>&=</code>	<code>x&=y</code>	<code>x = x & y;</code>
<code> =</code>	<code>x =y</code>	<code>x = x y;</code>
<code>^=</code>	<code>x^=y</code>	<code>x = x ^ y;</code>

All the advanced assignment operators, except for `+=`, will attempt to convert strings to numbers before performing the operation. If strings are used with the `+=` operator, the left operand is concatenated to the end of the right operand. For example, in Listing 2.10, the string "lighthouse" would be assigned to the variable `y` and the phrase "y= lighthouse" is written to the browser.

Listing 2.10 Using the Addition Operator to Perform String Concatenation

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Declare a string
y = "light";

//Concatenate the string "house" to the
//end of string stored in the variable y
y += "house";

// Print the output to the screen
document.write("y= ",y);

//-->
</script>
</html>
```

So far, you have only considered assignment of values to variables, but what about assigning a reference to a variable? When the assignment operator works on primitive values (numbers, strings, Boolean, null, and undefined), a copy of the value is made. When the assignment operator works on JavaScript objects, references to the objects are copied. To demonstrate this difference, Listing 2.11 creates a variable and an array to hold numbers. The variable is then copied by value to another variable, and the array is copied by reference to another array. To show the difference, the value stored in one of the variables and one of the arrays is changed, and then all the values are displayed in the browser as seen in Figure 2.6.

Listing 2.11 Assignment by Value Versus by Reference

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare a variable and an array object
var number1 = 94;
var arrayOfNum1 = new Array(23,86);

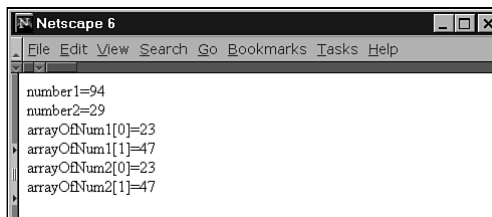
//Assign by value
var number2 = number1;

//Assign by reference
var arrayOfNum2 = arrayOfNum1;

//Modify value stored in copied variable and array.
number2 = 29;
arrayOfNum2[1] = 47;

//Display the values stored in each variable and array
document.write("number1=",number1,"<br>");
document.write("number2=",number2,"<br>");
document.write("arrayOfNum1[0]=",arrayOfNum1[0],"<br>");
document.write("arrayOfNum1[1]=",arrayOfNum1[1],"<br>");
document.write("arrayOfNum2[0]=",arrayOfNum2[0],"<br>");
document.write("arrayOfNum2[1]=",arrayOfNum2[1],"<br>");

//-->
</script>
</html>
```

**Figure 2.6**

The variables `number1` and `number2` are assigned values by value, whereas the arrays `arrayOfNum1` and `arrayOfNum2` are assigned values by reference.

Logical

JavaScript provides three logical operators. Without these operators, programs would be very long and complex. At first glance, most programmers might think that they

already know how these operators work, but it is important to dig a little deeper. Not knowing how the logical operators work can lead to what would appear to be random errors that are impossible to locate and correct. So take the time to read the description of each of the logical operators.

NOTE

JavaScript defines `true` as anything other than `0` (zero), `"` (empty string), `null`, `undefined`, or, of course, `false`.

Logical AND

The logical AND operator (`&&`) returns `true` if the expression to the left and the expression to the right of the operator evaluate to `true`. If either the left, right, or both expressions evaluate to `false`, the result of the operation is `false`.

Unfortunately, the implementation of the logical AND operator in JavaScript is more complex than what was just mentioned. The AND operation begins by evaluating the left operand. If the left operand evaluates `false`, the basic logic of the AND operator is complete, so the right operand is never evaluated. But if the left operand evaluates `true`, the right operand must be evaluated to determine the final result of the AND operation. In either case, the final result returned by the AND operation is actually the result of the last operand to be evaluated.

Logical OR

The logical OR operator (`||`) returns `true` if the expression to the left or the expression to the right of the operator evaluates to `true`. If both the left and the right expressions evaluate to `false`, the result of the operation is `false`.

Similar to the logical AND operator, it is important that you understand how JavaScript actually evaluates the logical OR operator. The OR operation begins by evaluating the left operand. If the left operand evaluates `true`, the basic logic of the OR operator is complete, so the right operand is never evaluated. But if the left operand evaluates `false`, the right operand must be evaluated to determine the final result of the OR operation. In either case, the final result returned by the OR operation is actually the result of the last operand to be evaluated.

Logical NOT

The logical NOT operator (`!`) is not as complex as the comparison operators. The result of the expression following the operator is inverted. If the expression evaluates to `true`, the result of the operation is `false`. If the expression evaluates to `false`, the result is `true`. When the expression evaluates to a non-Boolean value, it is converted to `true` or `false` before performing the inversion.

Comparison

JavaScript provides the usual comparison operators found in most languages plus a couple of more unusual ones. JavaScript had some inconsistencies in the operator

department during its early years, which resulted in the addition of some unconventional comparison operators. For this reason, it is important to understand how these operators work.

Equal

The equal operator (==) compares the value on the left of the operator to the value on the right of the operator. If the values are equal, `true` is returned from the operation. If the values are not equal, `false` is returned from the operation.

Originally, JavaScript attempted to convert the operands of the equality operator to the same type before performing a comparison. For example, if the left operand of an equal operator is a number and the right operand is a string, JavaScript would attempt to convert the string to a number so that two numbers are compared. In an attempt to guess what would be in the then unreleased ECMAScript standard, the decision was made not to do type conversion on the operands of the equality operator in JavaScript 1.2. When the ECMAScript standard was released, it supported type conversion, so JavaScript 1.3 came full circle by once again attempting to convert the operands of the equality operator to the same type before performing a comparison.

JavaScript determines which behavior should be used by the `<script>` tag. By setting the `language` attribute of the `<script>` tag equal to `JavaScript`, type conversion will be used. If the `language` attribute is set to `"JavaScript1.2"`, no type conversion will be used. An example of this behavior is demonstrated in the Listing 2.12.

Listing 2.12 Type Conversion and the Equal Operator

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
// Type-conversion turned on
document.write("The == operator with type-conversion turned on returns: ");
document.write(3=="3");
// -->
</script>

<script type="text/javascript1.2" language="JavaScript1.2">
<!--
// Type-conversion turned off
document.write("<br>The == operator with type- ");
document.write("conversion turned off returns: ");
document.write(3=="3");
// -->
</script>
</html>
```

TIP

Avoid setting the `language` attribute to `"JavaScript1.2"` in your code because the industry standard is for type conversion to be used on the operands of the equality operator.

So far you've seen when type conversion is used with the equality operator, but you haven't seen how the type conversion operates. Understanding how type conversions work for the equality operators will again save time when trying to find bugs. Type conversion adheres to the following rules:

- `True` is converted to the number 1, and `false` is converted to zero before being compared.
- If either of the operands are `NaN`, the equality operator returns `false`.
- `null` and `undefined` are equal.
- `null` and `undefined` are not equal to `0` (zero), `" "`, or `false`.
- If a string and a number are compared, attempt to convert the string to a number and then check for equality.
- If an object and a string are compared, attempt to convert the object to a string and then check for equality.
- If an object and a number are compared, attempt to convert the object to a number and then check for equality.
- If both operands of an equality operation are objects, the addresses of the two objects are checked for equality.

Not Equal

The not equal operator (`!=`) compares the value on the left of the operator to the value on the right. If the values are not equal, `true` is returned from operation. If they are equal, `false` is returned. The `!=` operator is victim to the same type-conversion bug as the `==` operator. As with the `==` operator, use the `language` attribute of the `<script>` tag to force the desired behavior.

Greater Than

The greater than operator (`>`) compares the value on the left of the operator to the value on the right. If the value on the left is greater than the value on the right, `true` is returned from operation. If the value on the left of the operator is less than or equal to the value on the right, `false` is returned. If either of the values is a string, it is converted to a number before the comparison takes place.

Less Than

The less than operator (`<`) compares the value on the left of the operator to the value on the right. If the value on the left is less than the value on the right, `true` is returned from operation. If the value on the left of the operator is greater than or equal to the value on

the right, `false` is returned. If either of the values is a string, it is converted to a number before the comparison takes place.

Greater Than or Equal

The greater than or equal operator (`>=`) compares the value on the left of the operator to the value on the right. If the value on the left is greater than or equal to the value to the right of the operator, `true` is returned from operation. If the value on the left of the operator is less than the value on the right, `false` is returned. If either of the values is a string, it is converted to a number before the comparison takes place.

Less Than or Equal

The less than or equal operator (`<=`) compares the value on the left of the operator to the value on the right. If the value on the left is less than or equal to the value on the right, `true` is returned from the operation. If the value on the left of the operator is greater than the value on the right, `false` is returned. If either of the values is a string, it is converted to a number before the comparison takes place.

Identity

The identity operator (`===`), also referred to as strict equal, compares the value on the left of the operator to the value on the right. If the value on the left is equal to the value on the right side of the operator, `true` is returned from operation. If the values are not equal, `false` is returned. No type conversion is performed on the operands before the comparison.

Non-Identity

The non-identity operator (`!==`), also referred to as strict does-not-equal, compares the value on the left of the operator to the value on the right. If the value on the left is not equal to the value on the right side of the operator, `true` is returned from operation. If the values are equal, `false` is returned. No type conversion is performed on the operands before the comparison is made.

NOTE

The identity and non-identity operators are only available in JavaScript 1.3 and later.

Conditional

Many programmers are not familiar with the conditional operator (`?:`), even though it exists in numerous languages. Most individuals will use the standard `if` statement rather than the conditional operator, even though they do the same thing. The conditional operator is a little harder to read than the standard `if` statement, but it is much more compact, which is important when download time is a consideration.

The format of the conditional operator can be a bit confusing. An expression that evaluates to a Boolean is always placed to the left of the question mark (`?`). If the expression evaluates to `true`, the value between the question mark and the colon (`:`) is

returned from the operation. If the expression evaluates to false, the value following the colon is returned. In Listing 2.13, a standard `if` statement is shown, along with same functionality produced by using the conditional operator. Figure 2.7 shows that the same functionality is produced from both the `if` statement and the conditional operator.

Listing 2.13 *The Conditional Operator and `if` Statement Are Compared*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Set the mail flag to "YES"
mailFlag = "YES"
var message1;
var message2;

//Standard if statement
if (mailFlag == "YES")
message1 = "You have email!";
else
message1 = "No email.";

//Same statement using conditional operator
message2 = (mailFlag == "YES") ? "You have email!" : "No email.";

// Print the message to the screen
document.write("The if statement returns: ",message1,"<br>");
document.write("The conditional operator returns: ",message2);

//-->
</script>
</html>
```

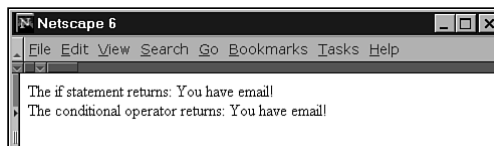


Figure 2.7

The result of using the `if` statement is the same as using the conditional operator.

Bitwise

The bitwise operators look similar to the comparison operators, but their functionality is very different. If you have ever worked with truth tables, you might recognize the operators in this section. Bitwise operators in JavaScript only work with integers that

are 32 bits in length. If an integer is not 32 bits, it is turned into a one because the bitwise operators evaluate numbers at the binary level where everything is ones and zeros. Bitwise operations are not used often in programming, but there are times when the operators are indispensable.

Bitwise AND

The bitwise AND operator (&) looks at the integer numbers on both sides of the operator as 32-bit binary numbers. Then the logical AND (&&) operator, discussed earlier in the chapter, individually evaluates each of the 32 bits representing the number to the left of the operator to the corresponding bit of the number to the right of the operator. The 32-bit binary result of logical AND operation is converted to an integer value and returned from the bitwise AND operation.

Bitwise OR

The bitwise OR operator (|) looks at the integer numbers on both sides of the operator as 32-bit binary numbers. Then the logical OR (||) operator, discussed earlier in the chapter, individually evaluates each of the 32 bits representing the number to the left of the operator to the corresponding bit of the number to the right of the operator. The 32-bit binary result of logical OR operation is converted to an integer value and returned from the bitwise OR operation.

Bitwise XOR (exclusive OR)

The bitwise XOR operator (^) looks at the integer numbers on both sides of the operator as 32-bit binary numbers. Unlike the bitwise OR operator, bitwise XOR uses a special version of the logical OR operator, called exclusive OR, to evaluate each bit of a binary number.



NOTE

An exclusive OR operation returns `true` if either the value to the left or the value to the right of the operator is `true`, but not both. If both values are `false` or both values are `true`, the result of the operation is `false`.

The exclusive OR individually evaluates each of the 32 bits representing the number to the left of the bitwise XOR operator to the corresponding bit of the number to the right of the operator. The 32-bit binary result of exclusive OR operation is converted to an integer value and returned from the bitwise XOR operation.

Bitwise NOT

The bitwise NOT operator (~) is simpler than the bitwise AND, OR, and XOR operators. The bitwise NOT operator begins by looking at the number to the right of the operator as a 32-bit binary number. Each bit of the given number is reversed so that all ones become zeros and all zeros become ones. The 32-bit binary result is converted to an integer value and returned from the bitwise NOT operation.

Shift Left

The shift left operator (<<) looks at the integer to the left of the operator as a 32-bit binary number. All the bits in this number are shifted to the left by the number of positions specified by the integer to the right of the operator. As the bits are shifted to the left, zeros are filled in on the right. Because the number can only be 32 bits long, the extra bits on the left are lost. The 32-bit binary result of shifting operation is converted to an integer value and returned from the shift left operation.

Shift Right with Sign

The shift right with sign operator (>>) is similar to the shift left operator. The shift right with sign operator looks at the integer to the left of the operator as a 32-bit binary number. All the bits in this number are shifted to the right by the number of positions specified by the integer to the right of the operator. As the bits are shifted to the right, either ones or zeros are filled in on the left. If the original number is positive, ones are added to the left side of the binary number. On the other hand, if the original number is negative, zeros are used. Because the result can only be 32 bits long, the extra bits on the right are lost. The 32-bit binary result of the shifting operation is converted to an integer value and returned from the shift right with sign operation.

Shift Right Zero Fill

The shift right zero fill operator (>>>) operates the same as the shift right with sign operator, except that the binary number is always padded on the left with zeros, regardless of the sign of the original integer.

Precedence

JavaScript, similar to other languages, enables numerous operators to be used in one expression. Because operators can appear just about anywhere within an expression, JavaScript follows guidelines that determine which operator is evaluated first, second, third, and so on. Table 2.7 shows the precedence of all the JavaScript operators. The **Read From...** column tells what order (left-to-right or right-to-left) operators of equal precedence are evaluated. It is possible to override the precedence of operators by using parentheses.

TIP

Use parentheses to ensure that your code operates like you expect and to make your code more readable.

Table 2.7 Operator Precedence

Precedence	Read From...	Operator	Operator Type
Highest	L to R	., []	Member access
	L to R	(), new	Call/ create instance
	R to L	++, --, +,	Pre/Post
		-, !, ~,	Increment/Decrement,

Table 2.7 Continued

Precedence	Read From...	Operator	Operator Type
			void, and so on. delete, typeof
	L to R	*, /, %	Multiplication, division, modulus
	L to R	+, -	Addition, Subtraction
	L to R	<<, >>, >>>	Bitwise shift
	L to R	<, <=, >, >=	Relational operations
	L to R	==, !=, ===, !==	Equality operations
	L to R	&	Bitwise AND
	L to R	^	Bitwise XOR
	L to R		Bitwise OR
	L to R	&&	Logical AND
	L to R		Logical OR
	R to L	?:	Conditional
	R to L	=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, =	Assignment operation
Lowest	L to R	,	Multiple evaluation

Loops and Conditionals

Loops and conditionals give programs the power to make decisions and perform tasks multiple times. JavaScript provides the standard conditionals and looping structures that are available in many computer languages. In fact, these structures were patterned after those found in C, C++, and Java, so if you have written code in any of these languages, you will find this section very straightforward.

Conditionals

Conditional statements enable programs to make decisions based on preset conditions that use the operators discussed earlier in the chapter.

if

The `if` statement is by far the most common conditional statement simply because it is simple and easy to use. The format of a simple `if` statement looks as follows:

```
if (expression)
    statement;
```

If the expression in parentheses evaluates to `true`, the statement is executed; otherwise, the statement is skipped. The statement to be executed can appear on the same line as

the `if` expression, but the code is usually easier to read if the statement appears on the next line as shown in the preceding pseudo code. If two or more lines of code are to be executed, curly braces `{}` must be used to designate what code belongs in the `if` statement.

Use the keyword `else` to extend the functionality of the basic `if` statement to provide other alternatives if the initial statement should fail. The format of an `if...else` combination resembles the following:

```
if (expression)
    statement1;
else
    statement2;
```

Now, if the expression evaluates to `true`, `statement1` is executed; otherwise, `statement2` is executed. Listing 2.14 demonstrates the use of `if` and `else` with a hotel occupancy example. When executed, the code returns the message "There are not enough rooms for 5 guests."

Listing 2.14 Basic `if...else` Structures

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var emptyRooms = 2; //Two people per room
var numberOfGuests = 5;

if (emptyRooms == 0)
    document.write("There are no rooms available.");
else
{
    if ((emptyRooms*2) >= numberOfGuests)
        document.write("There are enough rooms for ",numberOfGuests," guests.");
    else
        document.write("There are not enough rooms for ");
        document.write(numberOfGuests," guests.");
    }
}

//-->
</script>
</html>
```

Notice how Listing 2.14 used curly brackets `{}` to nest an `if...else` structure inside another `if...else` structure. Nesting gives programs more decision-making power, but this power comes at the cost of readability.

else...if

The `else...if` phrase is used in place of nested `if...else` structures to make code more readable. Each `else...if` phrase is followed by an expression enclosed in parentheses. Use as many `else...if` statements as needed. Use a final `else` statement to execute code when all other conditionals evaluate to `false`. Listing 2.15 has the same functionality as the code in Listing 2.14, but it uses the `else...if` structure. This code displays the phrase, "There are not enough rooms for 5 guests."

Listing 2.15 Making Nested if...else Statements More Readable with the else...if Phrase

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var emptyRooms = 2; //Two people per room
var numberOfGuests = 5;

if (emptyRooms == 0)
    document.write("There are no rooms available.");
else if ((emptyRooms*2) >= numberOfGuests)
    document.write("There are enough rooms for ",numberOfGuests," guests.");
else
    document.write("There are not enough rooms for ",numberOfGuests," guests.");

//-->
</script>
</html>
```

switch

JavaScript offers the `switch` statement as an alternative to using the `if...else` structure. The `switch` statement is especially useful when testing all the possible results of an expression. The format of a `switch` structure resembles the following:

```
switch (expression)
{
    case label1:
        statement1;
        break;
    case label2:
        statement2;
        break;
    default:
        statement3;
}
```

The `switch` statement begins by evaluating an expression placed between parentheses, very similar to the `if` statement. The result is compared to labels associated with case structures that follow the `switch` statement. If the result is equal to a label, the statement(s) in the corresponding case structure are executed. A default structure can be used at the end of a `switch` structure to catch results that do not match any of the case labels. Listing 2.16 gives an example of the `switch` structure.

Listing 2.16 *Using the switch Structure*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var color = "green";

//Display the color of the car based on the variable "color"
switch (color)
{
    case "red":
        document.write("The car is red.");
        break;
    case "blue":
        document.write("The car is blue.");
        break;
    case "green":
        document.write("The car is green.");
        break;
    default:
        document.write("The car is purple.");
}

//-->
switch
</script>
</html>
```

There are a few key points to note about the format of the `switch` structure in Listing 2.16. First, notice that a colon (`:`) always follows a label. Second, curly brackets `{}` are used to hold all the case structures together, but they are not used within a case structure, even when multiple statements are to be executed. Finally, the keyword `break` is used to break out of the entire `switch` statement after a match is found, thus preventing the default structure from being executed accidentally. The result of executing the code in Listing 2.16 is the string "The car is green." being displayed.

Loops

There are times when the same portion of code needs to be executed many times with slightly different values. Use loops that run until a condition is met to create this functionality.

for

The for loop is a structure that loops for a preset number of times. JavaScript uses the C and C++ for loop structure. This particular structure is flexible, which makes this type of loop very useful.

From a very high level, the for loop is made up of two parts: condition and statement. The condition portion of the structure determines how many times the loop repeats, whereas the statement executes every time the loop occurs.

The condition structure is contained within parentheses and is made up of three parts, each separated by a semicolon (;). The first part of the condition structure initializes a variable to a starting value. In most cases, the variable is declared within this section as well as initialized. The second part is the actual conditional statement that determines how many times the loop will be iterated. The third and final part determines how the variable, which was initialized in the first part, should be changed each time the loop is iterated. This third part gives the for loop its flexibility by causing the variable to be incremented, decremented, factored, or any other adjustment trick you can devise. The format of the for loop appears as follows:

```
for (initialize; condition; adjust)
{
    statement;
}
```

It is important to take time to think about how to implement for loops because it is easy to accidentally create an infinite loop. Specifically, make sure that the conditional will catch the adjusted variable at some point. In many cases, it is advantageous to use the variable in the statement portion of the for loop, but take care not to adjust the variable in such a way that an infinite loop is created. Listing 2.17 makes use of the for loop to create a multiplication table as shown in Figure 2.8.

Listing 2.17 Multiplication Table Using for Loop

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

document.write("<h2>Multiplication table for 4</h2>");
for (var aNum = 0; aNum <= 10; aNum++)
{
    document.write("4 X ",aNum," = ",4*aNum,"<br>");
}

//-->
</script>
</html>
```

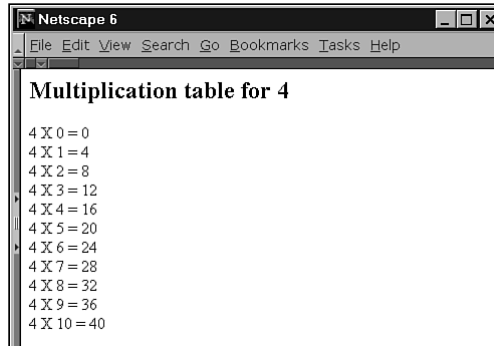



Figure 2.8

The multiplication table is created using a for loop.

while

When the for loop is too restrictive for a particular piece of code, consider using the while loop. The while loop can do everything that the for loop can do, but not as cleanly. So why even use the while loop? The while loop goes beyond the for loop's capabilities by not restricting the number of times the loop will execute.

The while loop is easy to understand if the phrase "While true, loop" is remembered. This phrase means that while the expression in parentheses evaluates to true, execute the statements in the loop. After the last statement in the loop is executed, go back to the top of the loop and evaluate the expression again. When the expression evaluates to false, the next line of code following the while loop structure is executed. To keep the loop from executing indefinitely, a statement must be included in the loop that modifies a variable in the expression. The format of the while loop resembles the following:

```
while (expression)
{
    statement;
}
```

Because the expression is evaluated before the loop, it is possible the loop will never be executed if the expression should evaluate to false the first time. Listing 2.18 simulates an automated traffic light using the while loop.

Listing 2.18 Automated Traffic Light Using while Loop

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var light = "red";           //traffic light
```

```
var counter = 1;           //create car traffic
var carsInLine = new Array(); //cars in line
//Make 5 cars go through intersection
while (counter <= 5)
{
    document.write("Car ",counter," approaches intersection.<br>");
    carsInLine[carsInLine.length++] = counter;

    //When 2 cars are in line light turns green
    if (carsInLine.length == 2)
    {
        light = "green";
        document.write("Traffic light turns ",light,"<br>");
    }

    //while light is green cars pass through intersection
    while (light == "green")
    {
        document.write("Car ",carsInLine[carsInLine.length-1]);
        carsInLine.length--;
        document.write(" goes through intersection.<br>");

        //When no cars are in line light turns red
        if (carsInLine.length == 0)
        {
            light = "red";
            document.write("Traffic light turns ",light,"<br>");
        }
    }
    counter++; //Next car
}

//-->
</script>
</html>
```

Listing 2.18 uses two while loops to simulate an automated traffic light. The first while loop could have just as easily been created using a for loop, but the second while loop would have been nearly impossible to implement using a for loop. The while loop handles this type of conditional loop with ease. In Figure 2.9, you see that the traffic light automatically turns green when two cars are in line at the intersection. After the two cars go through the intersection, the light turns red.

do...while

The do...while loop is simply a variation of the basic while loop that was just discussed. Other than syntax, the only difference between the do...while loop and the while loop is that the do...while loop always executes the loop once before evaluating the expression for the first time. This difference is seen in the following format:

```
do
{
    statement;
}
while (expression);
```

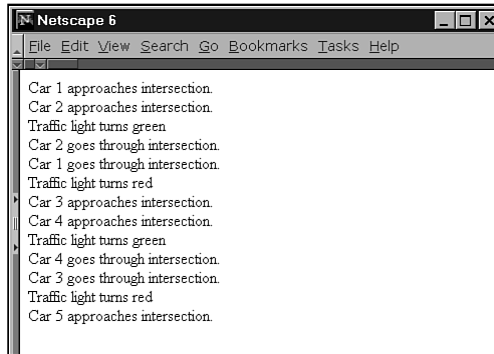


Figure 2.9

Simulating a traffic light using two while loops.

After the loop has executed for the first time, the expression in parentheses is evaluated. If true, the loop is executed again. When the expression evaluates to false, the next line of code following the while structure is executed. As was the case with the while loop, a statement must be included in the loop that modifies a variable in the expression to prevent an infinite loop. Also, notice that a semicolon (;) must be placed after the rightmost parenthesis. Listing 2.19 is the same automated traffic light simulator as shown in Listing 2.18, except `do...while` loops are used instead of while loops.

Listing 2.19 Automated Traffic Light Using `do...while` Loop

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var light = "red";           //traffic light
var counter = 1;            //create car traffic
var carsInLine = new Array(); //cars in line
//Make 5 cars go through intersection
while (counter <= 5)
{
    document.write("Car ",counter," approaches intersection.<br>");
    carsInLine[carsInLine.length++] = counter;
```

Listing 2.19 Continued

```
//If light green then execute loop more than once
do
{
    //When 2 cars are in line light turns green
    if (carsInLine.length == 2)
    {
        light = "green";
        document.write("Traffic light turns ",light,"<br>");
    }

    //When no cars are in line light turns red
    if (carsInLine.length == 0)
    {
        light = "red";
        document.write("Traffic light turns ",light,"<br>");
    }

    //Cars pass through intersection while light is green
    if (light == "green")
    {
        document.write("Car ",carsInLine[carsInLine.length-1]);
        carsInLine.length--;
        document.write(" goes through intersection.<br>");
    }
}
while (light == "green");

counter++;    //Next car
}

//-->
</script>
</html>
```

The output generated from running Listing 2.19 is exactly the same as executing Listing 2.18.

for...in

The `for...in` loop should not be confused with the `for` loop because they are quite different. The only similarity is that both iterate through the loop a set number of times, but this is as far as the similarity goes. The `for...in` loop is a special looping construct found only in JavaScript to provide access to all the enumerated properties of a JavaScript object. This includes elements of the Array object because they are stored the same way that property names are stored in JavaScript object. The statement(s) in the loop are executed for each property of an object until every property has been

accessed. Any parts of an object—such as methods and some properties—that are not enumerated are not accessed by this looping structure. The format of the `for...in` loop appears as follows:

```
for (variable in object)
{
    statement;
}
```

Before the statements in the loop are executed, a property name of the object specified to the right of the keyword `in` is assigned to the variable on the left side of the keyword `in`. The variable would then be used within the loop code. This process will continue until all the properties have been accessed. Unfortunately, the order in which the properties are accessed can vary, so do not assume a particular order. Listing 2.20 shows the properties of a simple HTML button.

Listing 2.20 *Accessing Property Names of the `Button` Object Using a `for...in` Loop*

```
<html>
<form name="aForm">
    <input type="button"
        name="Big_Button"
        value="Big Button"
        onClick="alert('The Big Button was pressed!')";
    >
</form>
<script type="text/javascript" language="JavaScript">
<!--

var aProperty;
for (aProperty in document.aForm.Big_Button)
{
    document.write(aProperty, "<br>");
}

//-->
</script>
</html>
```

Notice that in Figure 2.10, the name of the properties in the `Big Button` object, rather than the values stored in those properties, was returned.

break

The keyword `break` provides a way for JavaScript to exit out of loop structures and switch conditionals prematurely. Most of the time, the word `break` appears on a line by itself, but there are times when a label will follow the keyword. When a label is used, JavaScript completely breaks out of the area designated by `label` and proceeds to the code that follows the area.

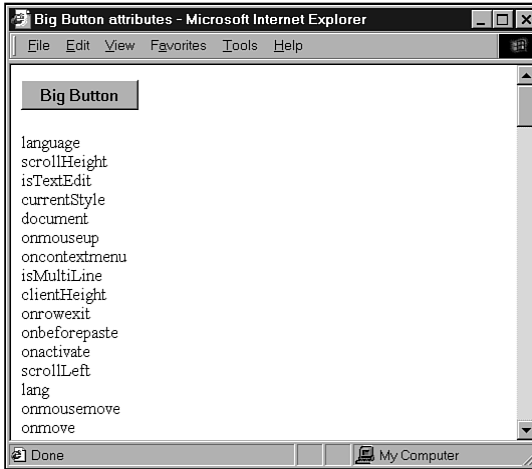


Figure 2.10

The properties of the Big Button object.

JavaScript labels can be thought of as placeholders. To label a statement, simply place the label name followed by a colon (:) in front of the code that needs to be broken out of during code execution. Labels are useful when working with nested loops, as shown in Listing 2.21.

Listing 2.21 Using Breaks and Labels

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Create outerloop
forLoop1:
for (var counter1 = 1; counter1 <= 5; counter1++)
{
    //Create innerloop
    for (var counter2 = 1; counter2 <= 5; counter2++)
    {
        //Display values in counters for both loops
        document.write("Counter1=",counter1);
        document.write(" Counter2=",counter2,"<br>");

        //Determine when to break out of loop
        if (counter2 == 3)
            break;
        if (counter1 == 3)
            break forLoop1;
    }
}
```

```
document.write("All done!");

//-->
</script>
</html>
```

Notice how the `break` statement with no label (see Figure 2.11) breaks out of just the inner loop. When the `break` statement is used with a label, JavaScript knows at which level to break.

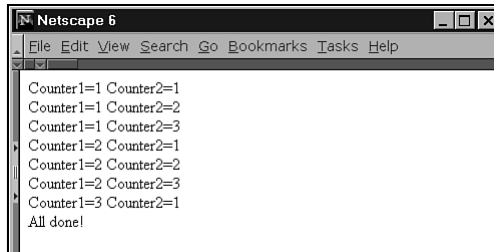


Figure 2.11

Result of using labels and nested loops.

continue

Unlike the JavaScript `break` structure, the `continue` statement forces the execution of the code to continue at the beginning of the loop. Similar to the keyword `break`, the `continue` keyword usually appears on a line by itself, but there are times when a label will follow the keyword. When a label is used, JavaScript immediately jumps to the beginning of the loop designated by a label and begins executing code.

The beginning of a loop varies depending on the type of loop structure. Table 2.8 shows where each looping structure jumps when a `continue` structure is encountered.

Table 2.8 *Where the continue Statement Jumps*

Looping Structure	Continue Jumps To
<code>for</code>	Expression in parentheses following the <code>for</code> keyword
<code>while</code>	Expression in parentheses following the <code>while</code> keyword
<code>do...while</code>	Expression in parentheses following the <code>while</code> keyword
<code>for...in</code>	Next property name in the object

CAUTION

A bug in Navigator 4 causes the expression in parentheses following the `while` keyword to not get executed when jumped to using a `continue` statement. Instead, execution of code starts at the top of the loop after the `continue` statement.

As discussed in the break section, JavaScript labels can be thought of as placeholders. To label a statement, simply place the label name followed by a colon (:) in front of the code where code execution should continue. Listing 2.22 demonstrates the use of label and continue.

Listing 2.22 Using the *continue* Statement and Labels

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Create outerloop
outerLoop:
  for (var counter1 = 1; counter1 <= 2; counter1++)
  {
    document.write("Top of outerLoop.<br>");
    //Create innerloop
    innerLoop:
      for (var counter2 = 1; counter2 <= 2; counter2++)
      {
        //Display values stored in counters of both loops
        document.write("Top of innerLoop.<br>");
        document.write("Counter1=",counter1,"<br>");
        document.write("Counter2=",counter2,"<br>");

        //Determine where to continue looping
        if (counter2 == 2)
        {
          document.write("Continue at top of innerLoop.<br>");
          continue;
        }
        if (counter1 == 2)
        {
          document.write("Continue at top of outerLoop.<br>");
          continue outerLoop;
        }
        document.write("Bottom of innerLoop.<br>");
      }
      document.write("Bottom of outerLoop.<br>");
    }

document.write("All done!");

//-->
</script>
</html>
```

This example is a bit complicated, so take time to compare Listing 2.22 to the output in Figure 2.12. Notice how the phrase "Bottom of innerLoop" was not printed after

the "Continue at top of innerLoop" because code execution jumped back to beginning of the innermost loop. When a label was attached to the continue keyword, code execution jumped back to the beginning of the loop labeled outerLoop.

CAUTION

Even though JavaScript provides labels, continue statements, and break statements, be careful when using these constructs because they can lead to errors that are hard to trace if they are used improperly.

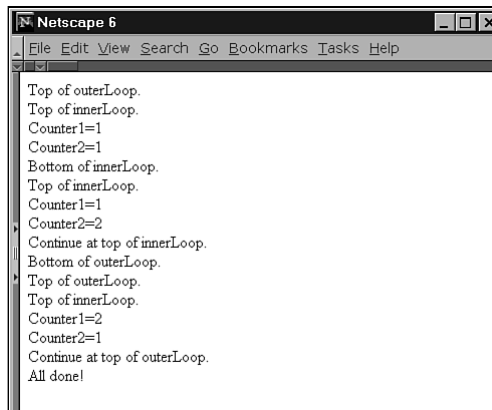


Figure 2.12

Result of using the continue statement in nested loops.

with

The object-oriented design of JavaScript quite often requires long lines of code to access properties and methods of objects. JavaScript provides a special with statement to help reduce the length of code needed to access these properties and methods. The with statement works by placing the repetitive portion of the object's path in parentheses after the with keyword. Now, any properties or methods used within the with statement will automatically have the repetitive portion of the object's path (located in parentheses) added to the front of the string. Listing 2.23 shows how the with statement can save time when resetting text fields to their default values.

Listing 2.23 *The with Statement Reduces Repetitive Code*

```
<html>
<!--Create a form that has 3 text fields and a reset button-->
<form name="personalInfoForm">
Name<input type="text" name="nameBox"><br>
Occupation<input type="text" name="occupationBox"><br>
```

Listing 2.23 Continued

```
Age<input type="text" name="ageBox"><br>
<input type="button" name="ResetButton"
value="Reset" onClick="ResetFields()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

//Set text field values initially
ResetFields();
//Reset text fields to default values
function ResetFields()
{
    with(document.personalInfoForm)
    {
        nameBox.value="[Enter your name]";
        occupationBox.value="Student";
        ageBox.value="";
    }
}

//-->
</script>
</html>
```

In Figure 2.13, you see that the text fields contain default data that appear initially as well as any time the Reset button is clicked. To achieve this functionality, the Reset button is connected to a function, called `ResetFields()`, that assigns default values to the text fields. To reduce repetitive code, the `with` statement was used in setting the default values, as seen in Listing 2.23.

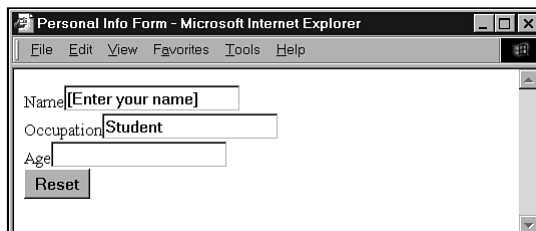


Figure 2.13

Using the `with` statement reduces the amount of code needed to set the default text box values.

Functions

One of the strengths of JavaScript is that it provides support for functions, which is uncommon among scripting languages. On the other hand, JavaScript functions are

not as fully developed as those found in languages such as C and C++. The functionality that JavaScript does provide through its functions is more than enough to make Web pages come alive.

Syntax

The syntax of JavaScript functions is very straightforward. All function declarations must begin with the keyword `function` followed by the name of the function. The name of the function is the name that will be used to call on the function within code. Parentheses are placed after the function name to hold arguments that are to be passed into the function. If more than one argument is to be passed into the function, use commas to separate the arguments. On the other hand, if no arguments need to be passed into the function, leave the space between the parentheses empty. Finally, curly brackets are used to contain the code related to the function. Curly brackets are not optional: They are required in JavaScript, even if the function is only made up of one line of code.

Call By Value Versus Call By Reference

If you have done programming in C or C++, you are probably familiar with the phrases *call by value* and *call by reference* as related to function arguments. In very basic terms, *call by reference* passes the location of the actual argument to the function, whereas *call by value* makes a copy of the argument to be used just within the function. JavaScript keeps the functionality of passing arguments simple by just using *call by value*. Using *call by value* gives the freedom to manipulate the arguments within the function without fear of changing the argument's values outside the function. Listing 2.24 shows an example of JavaScript *call by value*.

Listing 2.24 Call By Value

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Declare variables
var aString = "banana"
var aNumber = 15;

//Function declaration
function test(aString, aNumber)
{
    aString = "orange";
    aNumber = 124;

    //Display values stored in function variables.
    document.write("During function call:<br>");
    document.write("aStringCopy=",aString,"<br>");
    document.write("aNumberCopy=",aNumber,"<br>");
}
```

Listing 2.24 Continued

```
//Display variables before function call
document.write("Before function call:<br>");
document.write("aString=",aString,"<br>");
document.write("aNumber=",aNumber,"<br>");

//Call on function
test(aString,aNumber);

//Display variables after function call
document.write("After function call:<br>");
document.write("aString=",aString,"<br>");
document.write("aNumber=",aNumber,"<br>");

//-->
</script>
</html>
```

In Figure 2.14, you see that the values stored in `aString` and `aNumber` appeared to be changed while in the function `test`. But after exiting the function, the values reverted back to their initial value. What actually happened was that a local copy of the variables was made for use within the function. These new variables even have the same name as the ones that were passed into the function. After execution of the function was completed, the local variables no longer existed, so final values displayed were of the original variables.

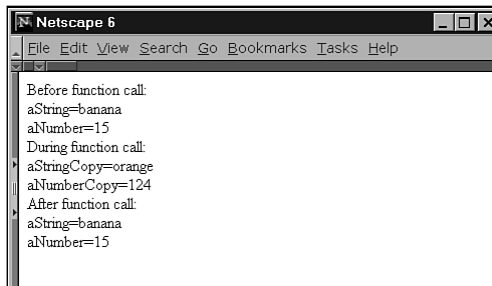


Figure 2.14

Result of using call by value.

If you need to pass in a large number of arguments but do not want to assign each one to a variable, it is possible to access the arguments as an array. To do this, leave the area in parentheses blank and use the `arguments` object and the array operator `[]` to access each argument. Listing 2.25 displays the arguments passed into the function using the `arguments` array as shown in Figure 2.15.

Listing 2.25 Accessing Function Arguments with the *arguments* Array

```

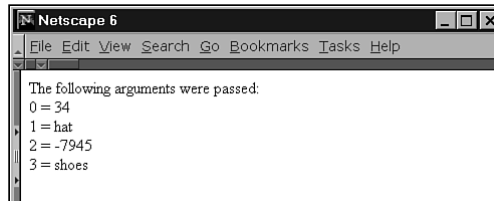
<html>
<script type="text/javascript1.2" language="JavaScript1.2">
<!--

//Create a function that displays the arguments it receives.
function displayArguments()
{
    document.write("The following arguments were passed:<br>");
    for(i=0; i<arguments.length; i++)
    {
        document.write(i, " = ",arguments[i], "<br>");
    }
}

//Pass some arguments into function
displayArguments(34,"hat",-7945,"shoes");

-->
</script>
</html>

```

**Figure 2.15**

The arguments passed into the `displayArguments()` function using the `arguments` array.

Returning Values

What if a value needs to be passed back from a function to be used later in the code? JavaScript provides a `return` statement that can be used in a function to return a value back to the statement that called the function. The value to be returned is simply placed after the keyword `return`. The undefined value is returned from a function if no value is specified after the keyword `return` or if the `return` statement is not used at all.

A value returned from a function can be assigned to a variable or used within an expression. In Listing 2.26, the value returned from the function is used in an expression to write the phrase "3*5=15" to the browser window.

Listing 2.26 Returning a Value from a Function

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Function declaration
function multiplyByFive(aNumber)
{
    return aNumber*5;
}

//Display variables before function call
document.write("3*5=",multiplyByFive(3));

//-->
</script>
</html>
```

Functions as Objects

The functions that have been described so far are created statically when the Web page is loaded, but there is also a dynamic function that is not created until it is called, which enables functions to be objects. The syntax for this type of function is very similar to the declaration of a variable.

```
var varName = new Function(argument1,...,lastArgument);
```

The keyword `Function` is used to create a new function dynamically from the arguments. All the arguments must be strings, and the last argument should always contain the functionality of the function. Listing 2.27 shows a simple example of using the `Function` object. Once again the phrase "3*5=15" is written to the browser window.

Listing 2.27 Returning a Value from a Function

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Create the dynamic Function
var multiplyByFive = new Function("y","return y*5");

document.write("3*5=",multiplyByFive(3));

//-->
</script>
</html>
```

Moving On

In this chapter, you were introduced to the details of the JavaScript language that are common across many computer languages. Armed with an understanding of JavaScript data types, variables, operators, loops, conditionals, functions, and basic syntax, which were covered in this chapter, you could start constructing JavaScript programs right now. But don't you want to learn some JavaScript specific programming techniques that will help you tap into JavaScript's full potential? In the next chapter, we will do just that by diving into some JavaScript programming techniques that take JavaScript beyond just a plain computer scripting language.



PART II

PROGRAMMING IN JAVASCRIPT

- 3 Programming Techniques **85**
- 4 Client-Side Scripting **107**
- 5 JavaScript on the Server-Side **157**
- 6 Windows Scripting **179**



CHAPTER 3

Programming Techniques

In Chapter 2, “Details of the Language,” you were given a quick overview of the core JavaScript language to show you differences and similarities to other computer languages. Now it is time to go to the next level by examining some of the features of the JavaScript language and programming techniques that will make your JavaScript programs more powerful and more resilient to errors and maybe even make you say, “Wow, I didn’t know JavaScript could do that!”

In this chapter we will look at pattern matching and event handling and show you how to use these two pieces of JavaScript functionality to make your JavaScript programs more powerful. This chapter will also examine two programming techniques that will make your JavaScript programs less error prone. You will be introduced to the `try...catch` block and shown how to use it. You will also be introduced to some free JavaScript debugging programs and techniques.

Pattern Matching

Two of the most common uses of Web pages today are for gathering and distributing data. These two areas, by nature, use data manipulation to understand what the user needs and then return information tailored to the user’s specifications. One of the key tools for processing data is pattern matching. Some scripting languages, such as Perl, are well suited to handle pattern matching, whereas others provide very little pattern matching capabilities. If you are familiar with Perl, you will probably recognize JavaScript pattern matching because it was based on Perl’s implementation of pattern matching. The implementation of pattern matching into JavaScript greatly aids in the processing of data for the Internet.

JavaScript uses the `RegExp` (short for Regular Expression) object to handle pattern matching. This object holds the pattern definition, as well as provides methods for performing matching. You'll begin by learning how to define patterns and then by learning how to use the `RegExp` objects to test for pattern matches.

Defining the `RegExp` Object

The `RegExp` object can be created in two different ways, which are similar to defining strings. The first way is to use the `RegExp` constructor and the keyword `new`:

```
var lastName = new RegExp("Jones");
```

This notation creates a new `RegExp` object called `lastName` and assigns the pattern *Jones*. The same functionality could have been accomplished by using a direct assignment:

```
var lastName = /Jones/;
```

To differentiate this notation from that used to define strings, the forward slash character (`/`) is used to designate the beginning and end of the pattern. Notice that forward slashes were not needed in the `RegExp()` constructor because this could be distinguished from the `String()` constructor.

Defining Patterns

The syntax used to define patterns in JavaScript could be considered a scripting language in itself because it is so extensive. There are special characters for creating almost any pattern one could imagine, including characters for handling groups, repetition, position, and so on. Table 3.1 shows the special pattern matching characters available in JavaScript.

Table 3.1 *Special Pattern Matching Characters*

Character	Description
<code>\w</code>	Matches any word character (alphanumeric).
<code>\W</code>	Matches any non-word character.
<code>\s</code>	Matches any whitespace character (tab, newline, carriage return, form feed, vertical tab).
<code>\S</code>	Matches any non-whitespace character.
<code>\d</code>	Matches any numerical digit.
<code>\D</code>	Matches any character that is not a number.
<code>[\b]</code>	Matches a backspace.
<code>.</code>	Matches any character except a newline.
<code>[...]</code>	Matches any one character within the brackets.
<code>^[...]</code>	Matches any one character not within the brackets.
<code>[x-y]</code>	Matches any character in the range of <code>x</code> to <code>y</code> .
<code>^[x-y]</code>	Matches any character not in the range of <code>x</code> to <code>y</code> .
<code>{x,y}</code>	Matches the previous item at least <code>x</code> times but not to exceed <code>y</code> times.

Character	Description
{x,}	Matches the previous item at least x times.
{x}	Matches the previous item exactly x times.
?	Matches the previous item once or not at all.
+	Matches the previous item at least once.
*	Matches the previous item any number of times or not at all.
	Matches the expression to the left or the right of the character.
(...)	Groups everything inside parentheses into a subpattern.
\x	Matches the same characters that resulted from the subpattern in group number x. Groups, which are designated with parentheses, are numbered from left to right.
^	Matches the beginning of the string or beginning of a line, in multiline matches.
\$	Matches the end of the string or end of a line, in multiline matches.
\b	Matches the position between a word character and a non-word character.
\B	Matches the position that is not between a word character and a non-word character.

These special pattern matching characters are used within the pattern to aid in defining complex patterns. Looking at Table 3.1, you might notice that characters such as the asterisk (*), plus sign (+), and backslash (\) hold special meanings that would keep them from being used as a literal. For example, what if you wanted to find all the plus signs (+) in a string? To use a literal plus sign, a backslash (\) must precede the sign. Table 3.2 shows all the characters that require a backslash character to be taken literally within a pattern.

Table 3.2 *Literal Characters*

Character	Description
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\v	Vertical tab
\/	Forward slash (/)
\\	Backward slash (\)
\.	Period (.)
*	Asterisk (*)
\+	Plus (+)
\?	Question Mark (?)
\	Horizontal bar ()

Table 3.2 Continued

Character	Description
\ (	Left parenthesis (
\)	Right parenthesis)
\ [	Left bracket ([
\]	Right bracket (])
\ {	Left curly brace {
\ }	Right curly brace }
\ XXX	ASCII character represented by the octal number xxx
\ xHH	ASCII character represented by the hexadecimal number HH
\ cX	The control character represented by x

There is one final piece of syntax that JavaScript provides for creating patterns. Unlike the syntax covered so far, these pieces of syntax appear outside the forward slashes that define the pattern. These attributes are shown in Table 3.3.

Table 3.3 Pattern Attributes

Character	Description
g	Global match. Finds all possible matches.
i	Makes matching not case sensitive.

Testing for Pattern Matches

When a pattern has been defined, it can be applied to a string by using special methods that exist in the `RegExp` and `String` objects. The pattern matching methods in the `String` object require `RegExp` objects, as shown in Table 3.4.

Table 3.4 Pattern Matching Methods in the String Object

Method	Description
<code>match(regExpObj)</code>	Searches for <i>regExpObj</i> pattern in string and returns result.
<code>replace(reqExpObj, str)</code>	Replaces all occurrences of the <i>regExpObj</i> pattern with <i>str</i> .
<code>search(reqExpObj)</code>	Returns the position of matching <i>regExpObj</i> pattern within the string.
<code>split(regExpObj, max)</code>	The string is split everywhere there is a matching <i>regExpObj</i> pattern up to <i>max</i> splits. The substrings are returned in an array.

The pattern matching methods in the `RegExp` object require `String` objects, as shown in Table 3.5.

Table 3.5 *Pattern Matching Methods in the RegExp Object*

Method	Description
<code>exec(str)</code>	Searches for pattern in <i>str</i> and returns result
<code>test(str)</code>	Searches for pattern in <i>str</i> and returns <code>true</code> if match found, otherwise <code>false</code> is returned
<code>(str)</code>	Same as <code>exec(str)</code> method

Listing 3.1 uses the `RegExp` constructor, special pattern syntax, and the `String` `replace()` method to replace digits 3 to 5 that exist in the string with nines as seen in Figure 3.1.

Listing 3.1 Using Regular Expressions

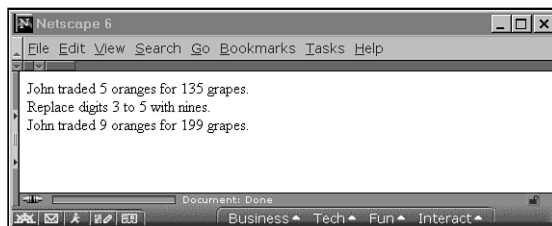
```
<html>
<script type="text/javascript" language="JavaScript">
<!--

//Create a text string
var str = "John traded 5 oranges for 135 grapes.<br>";

//Create RegExp object
var span3to5 = new RegExp("[3-5]", "g");

document.write(str);
document.write("Replace digits 3 to 5 with nines.<br>");
document.write(str.replace(span3to5, "9"));

//-->
</script>
</html>
```

**Figure 3.1**

The digits 3 to 5 are replaced with nines.

Events and Event Handlers

If you have ever done any graphical user interface programming, you are already familiar with the concept of events. *Events* are actions that take place outside of your program yet might or might not directly impact your program. Many events take place

within your browser that you never see and never need to worry about. But there are times when you really want to do something in your code when an event takes place.

For example, consider a Web page that provides users with an interface to change their password on some computer system. The Web page has a text box for the username and a text box for the new password as well as a button to submit the new password information. Before the username and password get submitted to some other system, you want to intercept the event that occurs when the user clicks the Submit button and verify that the password adheres to some specific password rules. A password rule might state that passwords must be greater than three characters in length but less than 15 characters. By having access to events that take place behind the scenes of the browser, you can cancel the submit action and alert the user to the problem with the password she entered.

In this section you will see the types of browser events that JavaScript will let you intercept and how to capture and take action based on those events.

Events

Many different types of events take place in browsers. JavaScript provides access to a number of events that would be useful to you, such as the `click` event, which occurs when the left mouse button is pressed once. Most of the events correspond to an HTML element that a user can see and manipulate such as a button or a check box. Table 3.6 contains a list of events that can be captured by JavaScript and the JavaScript object with which the event is associated.

Table 3.6 Browser Events

Event	Object	Occurs When...
Abort	Image	images do not finish loading
Blur	Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window	input focus is removed from the element
Change	FileUpload, Select, Text, Textarea	value of the element is changed
Click	Area, Button, Checkbox, Document, Link, Radio, Reset, Submit	element is clicked
Db1Click	Area, Document, Link	mouse button is double-clicked

Event	Object	Occurs When...
DragDrop	Window	object is dropped onto browser Window
Error	Image, Window	there is an error in the page or image
Focus	Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window	input focus is given to element
KeyDown	Document, Image, Link, Textarea	key is depressed
KeyPress	Document, Image, Link, Textarea	key pressed and held down
KeyUp	Document, Image, Link, Textarea	key is released
Load	Document, Image, Layer, Window	page is loaded into browser
MouseDown	Button, Document, Link	left mouse button is depressed
MouseMove	Window	mouse cursor is moved
MouseOut	Area, Layer, Link	mouse cursor is moved out of the element's bounds
MouseOver	Area, Layer, Link	mouse is moved over element
MouseUp	Button, Document, Link	mouse button is released
Move	Frame	element is moved
Reset	Form	form is reset
Resize	Frame, Window	browser window is resized
Select	Text, Textarea	input field is selected
Submit	Form	form is submitted
Unload	Window	current page is unloaded

Of the events covered in the previous table, the Error and Abort events deserve a little more explanation because they are not as straightforward as the rest.

The Error event is used by the Window and Image objects to indicate that an error occurred while either loading an HTML page or loading an image. This type of error will result in the browser issuing a JavaScript syntax error or a runtime error.

The Abort event is used by the Image object to indicate that the loading of an image was aborted. This type of event occurs often because users become impatient waiting for a large image to load, so they stop the image load before it completes by clicking the browser's Stop button or clicking a link to another page.

Event Handlers

Now that you know the types of events that JavaScript provides, access to them is just a matter of capturing those events. Events are captured using event handlers. By assigning a function or a single line of JavaScript code to an object's event handler, you can capture an event and take action. Table 3.7 shows all the event handlers and the events they are associated with.

Table 3.7 *Event Handlers*

Event	Event Handler
Abort	onAbort
Blur	onBlur
Change	onChange
Click	onClick
DblClick	onDbClick
DragDrop	onDragDrop
Error	onError
Focus	onFocus
KeyDown	onKeyDown
KeyPress	onKeyPress
KeyUp	onKeyUp
Load	onLoad
MouseDown	onMouseDown
MouseMove	onMouseMove
MouseOut	onMouseOut
MouseOver	onMouseOver
MouseUp	onMouseUp
Move	onMove
Reset	onReset
Resize	onResize
Select	onSelect
Submit	onSubmit
Unload	onUnload

Capturing Events

Event handlers can be defined in one of two ways. The first and most common way is to define the handler inside HTML tags much in the same way HTML tag properties are assigned. For example, to display an alert box when a button is clicked, simply assign a JavaScript alert box to the `onClick` event handler inside the button's HTML tag as follows:

```
<form name="myForm">
<input
  type="button"
  name="myButton"
```

```

    value="Press Me"
    onClick="alert('myButton was pressed')">
</form>

```

Anytime myButton is clicked an alert box will be displayed that tells the user that "myButton was pressed". Remember that not all events are associated with every object. To see what events and event handlers are available to a particular object, look for the object in Chapter 8, "Client-Side."

The second way to define event handlers is to define the handler inside JavaScript code using dot notation. Listing 3.2 demonstrates how to assign a JavaScript alert box to the onClick event handler using dot notation.

Listing 3.2 Defining Event Handlers Using Dot Notation

```

<html>
<form name="myForm">
<input type="button" name="myButton" value="Press Me">
</form>

<script type="text/javascript" language="JavaScript">
<!--
document.myForm.myButton.onclick="alert('myButton was pressed')";
//-->
</script>
</html>

```

In listing 3.2 myButton was initially created using standard HTML tags. Directly after creating the button JavaScript dot notation is used to access the button object and assign an alert box to the onclick handler.

NOTE

Notice that in Listing 3.2 the onclick property was written using a lowercase c rather than an uppercase C as was used when accessing the onClick property via the HTML input tag. This is not a typo! When defining event handlers inside HTML, use the uppercase characters as shown in Table 3.7. When defining event handlers inside JavaScript code using dot notation, the event handlers must not contain any uppercase characters.

Canceling Events

One of the most common uses of event handlers is validation of data entered through an HTML form. For example you might want to verify that a password entered by a user in a password change form is valid before submitting the form to the server. If the password entered by the user is not valid, the user should be notified of the problem and the form should not be submitted. Utilizing the material covered so far, it is easy to capture the Click event of the form's submit button and alert the user of the problems with the password entered. But how do you prevent the event from continuing and

the form from being submitted to the server? The Submit event can be canceled by simply returning `false` in the event handling routine. Listing 3.3 demonstrates how to cancel the form submission.

Listing 3.3 *Canceling the Submit Event*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function validatePassword()
{
    passwd = document.passwordForm.password.value;

    //Password must be between 3 and 15 characters
    if((passwd.length < 3) || (passwd.length > 15))
    {
        alert("Password must be less than 15 characters but greater than 3!");
        return(false);
    }
}

//-->
</script>

<center>
<h1>Password Change Page</h1>
Please enter your user name and new password.<br>
(Password must be between 3 and 15 characters.)<br><br>

<form name="passwordForm"
      action="success.html"
      onSubmit="return validatePassword()">
Username: <input type="text" name="username"><br>
Password: <input type="password" name="password"><br>
<input type="submit">
</form>
</html>
```

Not all the event handlers allow you to stop an event from taking place, but some do. Of the events that can be stopped, the value used to stop the event varies. Table 3.8 shows the events that acknowledge return codes and what values to return to cancel the event.

Table 3.8 *Event Handler Return Values*

Event	Value to Return to Cancel Event
OnClick	false
OnKeyDown	false

Event	Value to Return to Cancel Event
OnKeyPress	false
OnMouseDown	false
OnMouseOver	true (prevents URL from appearing in status bar)
onMouseUp	false
onReset	false
onSubmit	false

Invoking Event Handlers

There are times when you might want to explicitly invoke a particular event handler even though no event took place. This is easy to accomplish because the event handlers are essentially pointers to functions stored as a property of an object that should be executed when a particular event occurs. To invoke an event handler, simply use dot notation to execute the event handler as if it were a function. For example, in the following piece of code, we want to alert the user about a sweepstakes when he moves his cursor over the Lamborghini link. We also want to remind him of the sweepstakes when he goes back to the previous page. To do this, the event handler for the Lamborghini link is executed when the user clicks the Previous Page link.

```
<a href="sweepstakes.html"
  onMouseOver="alert('Enter our sweepstakes for a chance to win a brand new
↳sports car!')">Lamborghini</a><br>
<a href="intro.html"
  onClick="document.links[0].onmouseover()">Previous Page</a>
```

Timers

Even though JavaScript does not directly provide an event-driven timer, we will discuss timers in this section because timers should generally be thought of in terms of events. Because JavaScript does not directly provide a timer, it is possible to use the Window object's `setInterval()` method to serve the same purpose.

NOTE

The `setInterval()` method is supported in JavaScript 1.2 and higher.

The `setInterval()` method repeatedly calls a function or evaluates an expression each time a time interval (in milliseconds) has expired. This method continues to execute until the window is destroyed or the `clearInterval()` method is called.

For example, in Listing 3.4 the `setInterval()` method is executed when the document opens and begins to call the `dailyTask()` function every 20,000 milliseconds. The `dailyTask()` function evaluates the time each time it is called, and when it is 8:00 a.m., the code within the `if` statement is called, alerting the user and then clearing the interval. When the `clearInterval()` method is called, `setInterval()` halts execution.

Listing 3.4 Creating a Timed Event with the `setInterval()` Method

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
    function dailyTask()
    {
        var today = new Date();
        if ((today.getHours() == 8) && (today.getMinutes() == 0))
        {
            alert("It is 8:00 a.m.");
            clearInterval(timerID);
        }
    }
    //Set interval to 20,000 milliseconds
    timerID = setInterval("dailyTask()",20000);

//-->
</script>
</html>
```

As mentioned earlier the `setInterval()` method is only available in JavaScript 1.2 and higher. If you need to support an earlier version of JavaScript, you will have to use the `setTimeout()` method.

The `setTimeout()` method is usually used to evaluate an expression after a specific amount of time. Unlike the `setInterval()` method, the `setTimeout()` method is a one-time process that is not repeated an infinite number of times. Listing 3.5 produces the same result as Listing 3.4, using the `setTimeout()` method instead of the `setInterval()` method.

Listing 3.5 Creating a Timed Event with the `setTimeout()` Method

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
    function dailyTask()
    {
        var today = new Date();
        if ((today.getHours() == 8) && (today.getMinutes() == 0))
        {
            alert("It is 8:00 a.m.");
        }
    }
    //Set delayed execution of function to 20,000 milliseconds
    setTimeout("dailyTask()",20000);

//-->
</script>
</html>
```

Exceptions

As of JavaScript 1.4 exception handling has been added to allow you to capture and handle exceptions that occur in your JavaScript programs. The syntax that was chosen was the typical `try...catch`, `throw`, and `finally` statements that are found in other languages such as Java and C++. With these statements you can now throw your own errors, cleanly capture errors, and take the appropriate action based on the error thrown.

throw

The `throw` statement allows you to throw your own errors to be captured by the `try...catch` statement. The actual error that you throw can be any type of object so long as the `try...catch` statement knows how to handle the object. For example an error could be a number, string, or even a user-defined object that contains a number and an array of strings. The format of the `throw` statement resembles the following:

```
throw error
```

try...catch

The `try...catch` statement marks a block of code to try and a block of code to catch errors if an exception should be thrown. The format of the `try...catch` statement resembles the following:

```
try
{
    code
}
catch (error)
{
    code
}
```

The `try` block consists of one or more lines of code enclosed by brackets just below the `try` statement, whereas the `catch` block consists of one or more lines of code enclosed by brackets just below the `catch` statement. The `catch` block can also be passed the actual error that was thrown by specifying an object to hold the error in parentheses. This error object can then be used within the `catch` block. The error object and parentheses are optional and are only needed if you intended to access the actual error thrown from the `catch` block.

If an exception is thrown in the `try` block or within a function called by a statement in the `try` block using the `throw` statement, the code in the `catch` block is executed immediately to handle the error exception that was just thrown. If no exception is thrown while the `try` block is executed, the `catch` block is skipped.

NOTE

The `try...catch` statement can be nested to provide even more error handling.

In Listing 3.6, the `try...catch` and `throw` statements are used to help validate passwords in a simple password validation program. If a password is less than 5 characters in length or greater than 10, the user is presented with an error message thanks to the `try...catch` block.

Listing 3.6 Password Validation Using the `try...catch` Block

```
<html>
<script type="text/javascript" language="JavaScript">
<!-- hide

function ValidatePassword(password)
{
    try
    {
        //Make sure password has at least 5 characters
        if(password.length < 5 )
        {
            throw "SHORT";
        }

        //Make sure password has no more than 10 characters
        if(password.length > 10 )
        {
            throw "LONG"; //too many characters
        }

        //Password ok
        alert("Password Validated!");
    }
    catch(e)
    {
        if(e == "SHORT")
        {
            alert("Not enough characters in password!");
        }
        if(e == "LONG")
        {
            alert("Password contains too many characters!");
        }
    }
}
//-->
</script>

<h2>Password Validator</h2>

<form name="myform">
Please enter password: <input type="password" name="password"><br><br>
```



```
<input type=button name="validate"
      value="Validate!"
      onClick="ValidatePassword(myform.password.value)">
</form>

</html>
```

Runtime Errors

In addition to capturing user-defined errors, the `try...catch` block can also capture runtime errors that JavaScript throws without the assistance of the `throw` statement. You can capture these errors using the `try...catch` block much as you would capture your own user-created errors. The ECMA-262 (Third edition) standard defines six types of error objects that can be thrown by JavaScript. These errors are shown in Table 3.9 with a short description of the type of error they represent.

Table 3.9 Runtime Errors

Error Name	Description
<code>EvalError</code>	The <code>eval</code> function was used in a way that is not defined.
<code>RangeError</code>	A numerical value exceeded the allowable range.
<code>ReferenceError</code>	An invalid reference value was detected.
<code>SyntaxError</code>	A parsing error occurred.
<code>TypeError</code>	An operand was a type that was not expected.
<code>URIError</code>	One of the URI handling functions was used in a way that is not defined.

When a runtime error occurs, an `Error` object is returned. To determine the type of error, simply access the `name` property. Listing 3.7 purposely creates a runtime error by trying to use an undefined variable. The `catch` block determines the type error and displays an error message to the user.

Listing 3.7 Catching a Runtime Error

```
<html>
<script type="text/javascript" language="JavaScript">
<!-- hide
try
{
    //The following line will create a type error
    //because the variable aNum is undefined.
    sum = 5 + aNum;
    document.write("sum=",sum);
}
catch(e)
{
    if(e.name == "TypeError")
    {
```

Listing 3.7 Continued

```
        alert("A type error occurred.");
    }
}
//...>
</script>
</html>
```

finally

The `finally` block is an optional block of code that is executed each time an exception is thrown. This is especially useful in languages that work with files in which a file handle must be properly closed whether or not an error occurred. In the event that no error occurs, the code in the `finally` block is executed after the `try...catch` block but before the code following the `try...catch` block is executed. When an error is thrown, the `finally` block executes after the `try...catch` block. The format of the `try...catch` block plus the `finally` block looks as follows:

```
try
{
    code
}
catch (error)
{
    code
}
finally
{
    code
}
```

In Listing 3.8, the `finally` statement ensures that the password field is cleared whether or not the password is valid. If the password field had been cleared below the `finally` box, the field would only get cleared if the password was valid because there is a `return` statement in the `catch` block. If the password field had been cleared inside the `catch` box, the field would only get cleared if an error was thrown. Because the password field needed to be cleared regardless of errors, the `finally` block was used.

Listing 3.8 Password Validation Using the *finally* Block

```
<html>
<script type="text/javascript" language="JavaScript">
<!-- hide

function ValidatePassword(password)
{
    try
    {
        //Make sure password has at least 5 characters
        if(password.length < 5 )
```

```
{
    throw "SHORT";
}

//Make sure password has no more than 10 characters
if(password.length > 10 )
{
    throw "LONG"; //too many characters
}
}
catch(e)
{
    if(e == "SHORT")
    {
        alert("Not enough characters in password!");
    }
    if(e == "LONG")
    {
        alert("Password contains too many characters!");
    }
    return(1);
}
finally
{
    document.myform.password.value="";
}

//Password ok
alert("Password Ok!");
}
//-->
</script>

<h2>Password Validator</h2>

<form name="myform">
Please enter password: <input type="password" name="password"><br><br>
<input type="button"
    name="validate"
    value="Validate!"
    onClick="ValidatePassword(myform.password.value)">
</form>

</html>
```

Debugging

If you are used to doing programming in languages such as C++ and Java, you are probably accustomed to using some type of debugging tool to help you locate and fix

problems in your code. Now that JavaScript is finding its place in the coding community, we are finally seeing support of a native and comprehensive third-party scripting and debugging environment that is similar to those found in mature programming languages (such as C++, Visual Basic, and Java).

JavaScript scripting and debugging tools help the developers take advantage of more automated preventive and, to a lesser extent, corrective controls. If you do not have access to JavaScript debugging tools, or the debugging job is very small, a simple JavaScript alert box can work as a debugging tool. In this chapter, you will examine two free JavaScript debugging options that are at your disposal.

Microsoft Script Debugger

The *Microsoft Script Debugger (MSSD)* is a free downloadable script debugging tool that works as an integrated part of Internet Explorer (version 3.01 and later). MSSD also comes with Windows 2000 and Microsoft's Personal Web Server. You can use MSSD to write and, most importantly, debug your JavaScript (known as JScript with the Microsoft implementation) or Visual Basic Script (VBScript) code. MSSD has the advantage of being able to handle the debugging demands of ActiveX, Java, JScript, and VBScript. (The MSSD can be downloaded from <http://msdn.microsoft.com/scripting/>.)

Features

The following are the main features of the Microsoft Script Debugger:

- Dynamic view of HTML structure
- JavaScript, VBScript, and Java can be debugged seamlessly within the same document.
- Code is color coded
- Ability to set breakpoints
- Can step over, through, and out of each line of code
- Call stack
- Immediate expression window

Tips

The MSSD is easy to use, but there are a few things to know that will have you debugging your code quicker in MSSD.

Starting the Debugger

The only way to start MSSD is to first open Internet Explorer and load the desired HTML source file. Then you can activate MSSD by choosing View, Source.

NOTE

If MSSD isn't installed, viewing the source will open the source file in the Notepad editor.

To start the debugging process, choose Edit, Break at Next Statement from Internet Explorer, or choose Debug, Break at Next Statement from MSSD, and execute the script. This starts the debugger and stops it at the first statement in the current script.

The Break at Next Command

The Break at Next Statement command (which appears on the Script Debugger option of the View menu of Internet Explorer and the Debug menu of MSSD) is similar to a step command, in that the debugger executes the next statement in the script and then breaks, except that you can also use it when you are not currently running the script.

This is an important debugging feature of MSSD because a lot of JavaScript code is commonly declared in the header (or <head> tag) section of an HTML file, and this command is the only way to debug that code. This is because the code in the header of the file has already been executed by the time the HTML file is loaded. Also, any breakpoints set after the HTML file has been loaded are lost if you reload the page.

Evaluating Expressions

An expression can be evaluated with the aid of MSSD's immediate window and the following two methods:

- `Debug.write(string)`—This method writes a specified string, which is often the value of a variable, to the immediate window with no intervening spaces or characters between each string.
- `Debug.writeln([string])`—This method is identical to the preceding method, except that a newline character is inserted after each string. Also, the string argument is optional. If it's omitted, only a newline character is written to the immediate window.

A Final Word on the Microsoft Script Debugger

The Microsoft Script Debugger provides a very helpful environment for you to kick-start your JavaScript debugging and testing. The tools provided in MSSD are similar to tools usually found in full-blown programming language environments such as Visual Basic and C++. Also, MSSD's interface, setup, and installation are all very user friendly and intuitive.

However, MSSD has some limitations in that you need to switch frequently between Internet Explorer and MSSD to conduct debugging, and that you can't print source code. If you want more functionality than what MSSD offers, you might want to consider Microsoft's Visual InterDev 6.0. This product offers all the script-debugging features listed here, plus a full-blown Web developing environment.

At the least, MSSD is certainly a good tool to have in your arsenal, and it's a great value to boot because it's free. However, it can't take the place of writing solid code and systematically testing your code.

The `alert()` Method

If you have coded in any language for any length of time, you know that one of the simplest and quickest ways to debug functionality problems is to display the content of

important variables at various stages of your program's execution. By doing so, you can determine if your code is executing as you intended. This technique is especially useful when you cannot get your hands on a full-featured JavaScript debugger, or you are short on time and don't want to fire up a JavaScript debugger to solve a simple functionality problem. JavaScript has a handy method called `alert()` that lends itself well to stopping the execution of your script to see a value of a variable. With a little thought and proper placement of this method, you can quickly track down functionality problems in scripts.

Listing 3.9 contains some JavaScript code that is supposed to set the color and type of a car. One look at the result, as seen in Figure 3.2, and you can see something went wrong. The script was supposed to set the vehicle type to "car" with the `setType()` function and the vehicle color to "red" with the `setColor()` function. If these two functions had worked properly, a string would be written to the screen; otherwise, an alert message would be displayed to let you know that one of the assignment operations failed. No alert message was displayed, but the variable representing the vehicle's color was never set to "red".

Listing 3.9 Problem Code

```
<html>
<script type="text/javascript" language="JavaScript">

//Create two global variables used to describe the vehicles
var vehicleColor;
var vehicleType;

//Set the type of vehicle
function setType()
{
    return(vehicleType="car");
}

//Set the color of the vehicle
function setColor()
{
    return(vehicleColor="red");
}

//If the vehicle type and color were not properly set alert the user.
if(setType() || setColor())
{
    document.write("The " + vehicleType + " is " + vehicleColor);
}
else
    alert("The vehicle type and color could not be set");

</script>
</html>
```

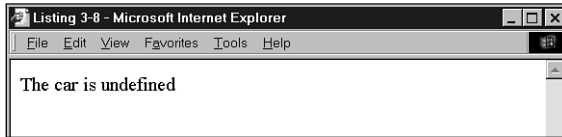


Figure 3.2

There is a problem associated with setting the color of the car.

In Listing 3.10, `alert()` methods are used to debug the code. An `alert()` method is placed in each of the variable-setting functions to determine if each function is being executed. More `alert()` methods are placed before and after the `if` statement to show how the conditional evaluates.

Listing 3.10 Debugging Using the `alert()` Method

```
<html>
<script type="text/javascript" language="JavaScript">

//Create two global variables used to describe the vehicles
var vehicleColor;
var vehicleType;

//Set the type of vehicle
function setType()
{
    alert("Inside the setType function."); //Debug statement
    return(vehicleType="car");
}

//Set the color of the vehicle
function setColor()
{
    alert("Inside the setColor function."); //Debug statement
    return(vehicleColor="red");
}

//Debug statement
alert("Before if statement: type="+vehicleType+" color="+vehicleColor);

//If the vehicle type and color were not properly set alert the user.
if(setType() || setColor())
{
    //Debug statement
    alert("After if statement: type="+vehicleType+" color="+vehicleColor);

    document.write("The " + vehicleType + " is " + vehicleColor);
}
```

Listing 3.10 Continued

```
else
    alert("The vehicle type and color could not be set");

</script>
</html>
```

When Listing 3.10 is executed, the first alert box displayed shows that both variables are undefined before the execution of the `if` statement. The next alert box shows that the `setType()` function was executed. The final alert box shows the vehicle type set to "car", but the color is still undefined after the `if` statement, as you can see in Figure 3.3. What happened to the `setColor()` function? The `alert()` method in the `setColor()` function was never executed, which lets us know that the `setColor()` function was never called from within the `if` statement.

**Figure 3.3**

Debugging using the `alert()` method.

If the first argument in a logical OR operation evaluates to `true`, the second argument is never evaluated. Because the `setType()` function returned `true`, the `setColor()` function was never executed. The problem is easily corrected by simply changing the logical OR operator to a logical AND operator.

CAUTION

After your code checks out, don't forget to remove the `alert()` methods that you added for debugging or else your program will display these alerts to those who use your scripts.

Moving On

In this chapter, you were introduced to some programming techniques that will make your JavaScript programs more powerful and more resilient through the use of JavaScript pattern matching, event handling, exception handling, and debugging. Part II, "Programming in JavaScript," provides a look at the environments in which JavaScript can be interpreted and into the actual use of the language.



CHAPTER 4

Client-Side Scripting

The first three chapters of this book introduced you to JavaScript, exposed you to its semantics, and covered several programming techniques to help you get started. In Part II, “Programming in JavaScript,” we are going to cover the environments in which JavaScript is interpreted, including the client-side, server-side, and in the Windows Script Host environment.

In this chapter specifically, you will look at several aspects of the language. We’ll look at how the browsers interpret JavaScript, how to deal with some of the issues that surround browsers, and how they handle the scripts—issues such as bugs and differences in functionality.

Later, you will look at how JavaScript can extend a browser’s functionality. This is accomplished with some of its object arrays and through the use of LiveConnect—the Java to JavaScript bridge. We will also cover tasks that will allow you to process form data and without having to send pages back and forth to the server. Cookies, another key component, will also be covered in this chapter.

The next topic we will discuss is Window manipulation. Windows appear to be very basic on the outside, but the underlying JavaScript provides a lot of power for configuring and manipulating windows to meet your design needs.

This chapter will also include coverage of the dynamic positioning of HTML and XHTML elements and their manipulation using JavaScript. Finally, the chapter will take a look into XUL—the XML-Based User Interface Language, which is a new language created by our friends at Mozilla.org and used to build the user interface for the Netscape 6 and Mozilla browsers.

This chapter has a lot of material in it, so let's get started by talking about which browsers support JavaScript and some of their issues.

Supporting Browsers and Their Issues

Even though JavaScript seems to have been a foundation building block in today's Internet technology, it wasn't always there. It wasn't until Netscape Navigator 2.0 and Internet Explorer 3.0, that its functionality was included. Currently, only four major browsers interpret JavaScript: Netscape Navigator, Internet Explorer, Opera, and HotJava. However, just because they interpret the language does not mean that they do so in the same manner, which is something you will be looking into shortly.



NOTE

Because many users use AOL's online service to connect to the Internet, you should know that the AOL browser is actually an Internet Explorer browser with different "chrome."

For those of you familiar with Java, you know that various *JVMs*, or *Java Virtual Machines*, can interpret the language differently. JavaScript is similar, except that the market can often be considered more fragmented. For the most part, browsers do interpret JavaScript the same—at least the core language semantics. The big distinction is that they all have different version support as well as diverse bugs.

To give you an idea of the versions and support among these browsers, take a look at Table 4.1. It breaks down these runtime environments by browser version and language version.

Table 4.1 *Language Support by Browser Version*

Browser	Version	Language
Netscape Navigator	2.0	JavaScript 1.0
	2.02 for OS/2	JavaScript 1.1
	3.0	JavaScript 1.1
	4.0–4.05	JavaScript 1.2
	4.06–4.5	JavaScript 1.3
	6.0	JavaScript 1.5
Microsoft Internet Explorer	3.0	JScript 1.0
	4.0–4.5	JScript 3.0
	5.0	JScript 5.0
	5.5	JScript 5.5
	6.0	JScript 6.0
HotJava	3.0	JavaScript 1.4
Opera	3.0–3.5	JavaScript 1.1
	4.0–5.0	JavaScript 1.3
		JavaScript 1.4

NOTE

Opera has done an excellent job of keeping up with standards in its browsers; however, understanding what it supports exactly can be a bit confusing. Table 4.1 is a rough idea of their support, but for more information we recommend that you visit <http://www.opera.com/opera4/specs.html#ecmascript> for version 4 information and <http://www.opera.com/opera5/specs.html#ecmascript> for version 5.

JAVASCRIPT 1.1 IN NAVIGATOR 2.02 FOR OS/2

If you take a second look at Table 4.1, you will notice that the 2.02 version of Navigator for OS/2 is JavaScript 1.1-compliant. This is because the OS/2 versions of the Navigator browsers were co-developed by engineers from both Netscape and IBM.

The OS/2 version of Navigator was not announced or in beta until the 3.0 version of the browser was already released. Those of you who used the 3.0 browsers heavily might have noticed that they seemed more buggy than the 2.0 versions. Added enhancements seemed to make the code more unstable, which prompted the OS/2 version using the more stable 2.0 interface to be built, but included the 3.0 backend.

Because browser versions are often determined using JavaScript, an OS/2 user with Navigator 2.02 can start the browser with the “-3” option that will tell its user agent string to report that it is a 3.0 browser rather than a 2.02. Because it has the capability to interpret all the JavaScript on the page, this option allows the user to experience all the enhancements for the 3.0 browsers.

These facts are important for programmers to properly implement features in their scripts. You can now check the Navigator 2.02 for OS/2 browser and know that it is JavaScript 1.1-compliant—with or without the -3 option. All of this, of course, was remedied with the release of Netscape Communicator 4.61 for OS/2, which has support for JavaScript 1.3.

New browsers are released a couple of times a year, which forces the market to become fragmented in terms of the JavaScript runtime environment with which users will be accessing a site. Extra care needs to be taken in writing scripts so browsers will interpret them correctly. As discussed in Chapter 2, “Details of the Language,” most of the version control can be accomplished using the language or type attribute of the `<script>` element, but not all browsers correctly implement this. As a programmer, you will have to write in code to accommodate these browsers.

The first browser you are going to look at is Netscape’s Navigator browser. It was the pioneering browser that first interpreted the language co-developed by Sun and Netscape. We will also look at Internet Explorer, Opera, and HotJava as well.

Netscape Navigator

Netscape Navigator (see Netscape 6.0 in Figure 4.1) first included its support for JavaScript in its version 2 browsers. This was a major step for JavaScript because Navigator was by far the most widely used browser in the world. Web developers could add scripts to their pages and feel very confident that the majority of their visitors would be able to experience their enhancements. And things have not stopped there.

Recently, Netscape released version 6 of its browser, which was a complete rewrite and is shown in Figure 4.1, from the ground up. This time, its support for official standards has been relatively unmatched. It has even gone so far as to not support old elements and tags that were implemented in Navigator 4, which provides its own problems, but reflects Netscape's dedication to standardization.

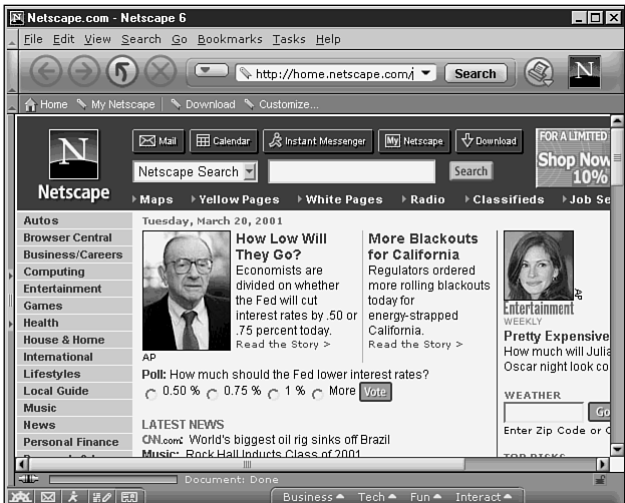


Figure 4.1
Netscape 6 browser.

Similar to anything else Netscape has implemented in its browsers, it has continued to expand the language and add to it. With each major release, new functionality through standards support have been added that keep it ahead of other browsers. Table 4.2 outlines the JavaScript support in these browsers and gives you some additional information.

Table 4.2 JavaScript Language Support by Browser

Browser Version	Language Version	Notes
2.0	JavaScript 1.0	First browser to interpret JavaScript.

Browser Version	Language Version	Notes
2.02 for OS/2	JavaScript 1.1	As discussed in previous sidebar, this browser has the 2.0 interface, and it has the 3.0 back-end for rendering HTML and interpreting JavaScript.
3.0	JavaScript 1.1	First version to support the <code>src</code> attribute of the <code><script></code> tag, which can be used to include external JavaScript source.
4.0–4.05	JavaScript 1.2	Enhancements for Dynamic HTML and added signed scripts as a model of security.
4.06–4.7	JavaScript 1.3	Completes Navigator's support for ECMAScript 1st Edition, as well as other enhancements expected in the 2nd Edition.
6.0	JavaScript 1.5	Adds support for the DOM Level 1 and some Level 2 as well as ECMAScript 3rd Edition.

TIP

Netscape's DevEdge site has more information on JavaScript bugs. It can be accessed at <http://developer.netscape.com/tech/javascript/index.html>, and then follow the link for Known Bugs.

Internet Explorer

Even though Internet Explorer (see version 5.5 in Figure 4.2) was the second browser to follow suit in its support of JavaScript, it seems to have avoided the Navigator problems just discussed. However, JScript does have issues that inhibit its performance and functionality as well.

Because Microsoft did not want to purchase the licensing to JavaScript from Netscape to implement in its Internet Explorer browser, it had to reverse-engineer the scripting language and give it a new name. This led to the birth of JScript.

In the short run, this seemed like a bad thing for JavaScript programmers. Now that two versions of the language were on the market, how could programmers be assured that their scripts would work in both browsers? Luckily, this version incompatibility only lasted for version 3 of Internet Explorer. By the time version 4 of the browser came out, the first edition of the ECMAScript standard was well on its way to adoption, and Microsoft based its JScript 3.0 on the standard. This refocused and aligned both JavaScript and JScript and reduced the problems of incompatibility between them. There are, however, a few quick things to watch for in the older versions of the browser:

- Microsoft provides the ability for users to update their scripting engines without updating the browser. It also distributes different engines with minor updates to the browser.
- No support for the Image object on the Windows platform in version 3 of the browser.
- Support for the src attribute of the <script> tag was implemented in the 3.02 maintenance release of Internet Explorer and did not fully work.



Figure 4.2
Internet Explorer 5.5 browser.

NOTE

Microsoft refers to its interpreter as a *JScript Scripting Engine*. This terminology will be used in this section.

Table 4.3 breaks down the browsers that have been released by Microsoft and its JScript support by version. The table also includes some notes that give more information about what the release added to the language.

Table 4.3 JScript Language Support by Browser

Browser Version	Language Version	Notes
3.0	JScript 1.0	First Internet Explorer browser to interpret JScript.
4.0–4.5	JScript 3.0	Added enhancements for Dynamic HTML and support for

Browser Version	Language Version	Notes
		the first edition of ECMAScript. This is the first version to fully support the <code>src</code> attribute of the <code><script></code> tag, which can be used to include external JavaScript source.
5.0	JScript 5.0	Supports the third edition of ECMAScript.
5.5	JScript 5.5	Supports ECMAScript 3rd Edition and DOM Level 1 and some Level 2 items.

There is another topic that we want to discuss here, which is the user's ability to update his scripting engine without having to update his browser. This makes it very possible for a user to implement an Internet Explorer 5.0 browser that has the 5.5. JScript engine.

One good thing is that the more recent versions of the engines have functions to allow developers to determine what version of the engine users are running. Watch out because using some of these functions on version 3.0 engines causes errors. Luckily, version 3 is few and far between at this point, so it puts developers at less of an impossible situation.

Listing 4.1 includes the Microsoft-specific elements that a developer can use to determine the version of the scripting engine the user has.

Listing 4.1 JScript's Elements for Determining Scripting Engine Version Information

```
<script language="JScript" type="text/javascript">
<!--
// Create a variable to hold all the engine information in a single string.
var jscriptVer = ScriptEngine() + " " + ScriptEngineMajorVersion() + "." +
    ScriptEngineMinorVersion() + " Build " +
    ScriptEngineBuildVersion();

// Write the string to the user's browser.
document.write('<b>You are running:</b> ' + jscriptVer);

// Write each of the individual elements of the engine's version to the
// browser.
document.write('<br><br><b>ScriptEngine:</b> ' + ScriptEngine());
document.write('<br><b>ScriptEngineMajorVersion:</b> ');
document.write(ScriptEngineMajorVersion());
document.write('<br><b>ScriptEngineMinorVersion:</b> ');
document.write(ScriptEngineMinorVersion());
```

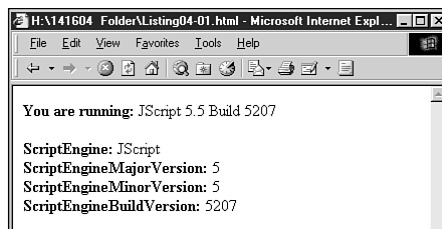
Listing 4.1 Continued

```
document.write('<br><b>ScriptEngineBuildVersion:</b> ');  
document.write(ScriptEngineBuildVersion());  
// -->  
</script>
```

TIP

In case you are wondering, the scripting engine for JScript is contained in the `jscript.dll`. You can check the properties in this DLL file to see what version you have on your machine.

The result of opening Listing 4.1 in Internet Explorer 5.5 on Windows is shown in Figure 4.3. As you can see, this can be very helpful if developers need to know the specific build the user implements when executing his scripts. However, you should be careful to avoid versions that do not support these elements.

**Figure 4.3**

Internet Explorer 5.5 displaying its scripting engine information.

Opera

The Opera browser (see version 5.0 in Figure 4.4) has received a lot of press because of its support for standards, such as *cascading style sheets (CSS)*. The developers at Opera Software have taken on the responsibility of providing a lot of functionality to their users without a lot of unneeded flash. Opera is mostly a pure rendering engine, but it does have support for Navigator plug-ins and Java. This allows its users to gain the needed support to position it as a real alternative.

The main thing you will want to watch out for in Opera browsers is the user agent string—it has the ability for users to select the string sent. This was included to allow the Opera browser to access pages built for Navigator 4, or perhaps Internet Explorer 5, which can make it tricky for JavaScript programmers to accurately detect the browser version.



Figure 4.4

The Opera 5.0 browser is available at <http://www.opera.com>.

HotJava

The final browser that we are going to mention is Sun's HotJava browser, which included support for JavaScript in version 3.0. It uses the Rhino implementation (JavaScript engine written in Java) from Mozilla.org for its interpretation and is the only browser that was officially released that supported JavaScript 1.4. To be honest, this browser is rarely used and has not been updated in over a year, but we did want to mention it.

Extended Browser Functionality

In addition to the scripting functionality you can add to Web pages, JavaScript can also be used to extend your browser's functionality. Today's most functional browsers support plug-ins or ActiveX controls. These plug-ins are loaded by the browser when it starts and are accessed when called via HTML tags on a page. Because plug-in instances run as objects within the browser's environment, some core JavaScript objects can be used to access, modify, and control their functionality.

JavaScript also supports interaction with Java applets embedded in a page. This type of interaction can occur in both directions—from the applet to scripts on a page, and from scripts to the applet. The technology behind this is referred to as LiveConnect and will be discussed in more detail later in the chapter.

Built-In Functions for Control

Not all browsers have the same methods for their extended functionality, but some common language elements allow developers to access these various items. These are considered *built-in* control elements in the JavaScript language itself, and access to this functionality is through two included arrays.

These two items are arrays containing references to all the applets the browser has loaded on a given page and all the plug-ins the browser has installed. Through these arrays, a developer is able to access and interact with these components.

Applet Array

The applet array stores references to all the Java applets loaded on a page. This allows a developer to retrieve the total number of applets on a page and to directly interact with them. If the developer is using LiveConnect to interact with the applet, the applet itself can be referenced by its indexed location. The second applet, for instance, could have its information accessed by the following JavaScript code:

```
var myAppletInfo = document.applets[1];
```

This allows developers to store information about this applet in a variable, where they can then parse it and process the information about the applet. The information stored in the `myAppletInfo` example would contain the applet's name, dimensions, and the layout package used to create the applet.

With the increased use of Web-based, enterprise-level applications, it is not uncommon for a developer not to know how many applets are on a page. Pages are built dynamically depending on the type of request by the user, and the resulting page might depend completely on the number of items the user selected to view. Because this number can vary, a developer might have to include JavaScript code that analyzes the page to see how many applets are on it. Listing 4.2 shows an example of this.

Listing 4.2 *Checking for Java Applets with the `length` Property of the Applet Array*

```
<html>
<head>
<title>Checking for Java Applets</title>
<script language="JavaScript1.1" type="text/javascript">
<!--
function checkApplets(){

    // Store the number of applets on the page in a variable
    var numApplets = document.applets.length;

    // If there are no applets on the page, then go to the "noapplets"
    // page.
    if(numApplets == 0){
        window.location.href = "http://www.purejavascript.com/noapplets.html";
```

```

// If there is only one applet on the page, go to the
// "oneapplet" page.
} else if (numApplets == 1){
    window.location.href = "http://www.purejavascript.com/oneapplet.html";

// If there are more than two applets on the page, go to the
// "manyapplets" page.
} else if (numApplets > 2){
    window.location.href = "http://www.purejavascript.com/manyapplets.html";
}
}
// -->
</script>
</head>

<body onLoad='checkApplets() '>
<center>
<table border="1">
    <tr>
        <td>
            <applet code="TestApplet"
                codebase="."
                width="200"
                height="35"
                name="AppletOne"
                mayscript>
                This browser is not able to run Java applets
                <param name="clickurl"
                    value="http://www.purejavascript.com/cgi-bin/redirect.cgi">
            </applet>
        </td>
    </tr>
    <tr>
        <td>
            <applet code="TestApplet"
                codebase="."
                width="200"
                height="35"
                name="AppletTwo">
                This browser is not able to run Java applets
                <param name="clickurl"
                    value="http://www.purejavascript.com/cgi-bin/redirect2.cgi">
            </applet>
        </td>
    </tr>
</table>
</center>
</body>
</html>

```

As you can see in Listing 4.2, after the page has fully loaded, an `onLoad` event handler is called by the `<body>` tag. This event triggers a function that checks to see how many applets were loaded on the page. Depending on the number of applets loaded, the browser might be redirected to another location. If the number of applets on the page is two, the page does not redirect to another location.

NOTE

You are only able to access properties and the results of applet methods after the applet is fully loaded and running. To ensure this, use the `onLoad` event handler in the `<body>` tag. Note that there are bugs in certain platforms of Navigator browsers that incorrectly fire this event. Be sure to account for these instances.

As you can see, the applet array can be a very helpful resource in accessing the applets on your pages. You should use it with caution, however, because not all browsers that support JavaScript support the ability to run applets.

Plug-In Array

The plug-in array is the second array that can be used to control elements loaded by an HTML page. Developers commonly determine whether a particular plug-in is installed on the user's machine by using this array. When this has been determined, the developer can then make the appropriate decisions about whether to try and load the plug-in.

The actual plug-in array has several properties that can be used to retrieve this information. These include items such as the name of the plug-in, the actual filename, and a description. Listing 4.3 demonstrates the use of the plug-in array by writing the information it can retrieve to the page. Figure 4.5 shows the result of running this in a browser with several plug-ins installed.

NOTE

The following example only works in Netscape browsers, but Internet Explorer does in fact support this array (or as Microsoft calls it, "collection"). For more information on what properties and methods are supported under this collection, visit http://msdn.microsoft.com/workshop/author/dhtml/reference/collections/plugins_0.asp.

Listing 4.3 Checking the Plug-Ins Array

```
<html>
<head>
<title>Checking the plug-ins Array</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
<!--
```

```
// Store the number of plug-ins in a variable.
var numPlugins = navigator.plugins.length;

// Write the title of the page.
if (numPlugins > 0){
    document.write('<h3>The Plug-ins You Have Installed</h3><hr>');
}else{
    document.write('<h3>You have No Plug-ins Installed.</h3>');
}

// Write the various installed plug-in information to the page.
for (i = 0; i < numPlugins; i++){
    currPlugin = navigator.plugins[i];
    document.write('<p><b>Name:</b> ' + currPlugin.name + '<br>');
    document.write('<b>Filename:</b> ' + currPlugin.filename + '<br>');
    document.write('<b>Description:</b> ' + currPlugin.description + '<br>');
}
// -->
</script>
</body>
</html>
```



Figure 4.5

The result of running Listing 4.3 in a browser with several plug-ins installed.

In the script, you access the total number of plug-ins installed by using the `length` property of the plug-ins array. After this has been done, determine the appropriate header to write to the page and run through a `for` loop writing the plug-ins information as well. This is a fairly simple example, but accessing these properties can be very

useful when determining what elements you want your pages to use when loaded by a browser.

LiveConnect

LiveConnect is a Netscape-specific technology, starting with Netscape Navigator 3, that provides a communication link among JavaScript scripts, Java, and plug-ins. It allows JavaScript to access certain Java core functionality through the `Packages` object. It can also access specific Java applets through the `Applet` array and plug-ins through the `Plugin` array.

In this section, you'll take a closer look at LiveConnect as it pertains to Java applets. You'll also step through some examples of using LiveConnect to extend the browser's interpretation of scripts and Java.

Why LiveConnect?

LiveConnect plays a very important role in linking JavaScript and Java together so that each can rely and expand on the functionality the other has to offer. The ability to dynamically change and interact with a Java applet on a page after it is loaded makes it easy for a Web developer to harness the power of Java. This can be done in modular fashion because it is completely possible that another developer wrote the applet.

On the flip side, LiveConnect enables an applet to access information contained in scripts. Because information can be passed to and from the applets, developers are able to maximize the functionality of their pages when using it. Using LiveConnect within an enterprise gives the developer the ability to exploit these features. Netscape has developed LiveConnect so that it fits nicely within its vision and adds to the mix of technologies that support the “write once, run anywhere” theories.

LiveConnect Examples

The following two examples will give you a better understanding of how LiveConnect works within Netscape's browsers. The first example shows you how JavaScript can access some of the Java language functionality natively (through the `Packages` object). The second example shows how you can use JavaScript to interact with an applet.

Listing 4.4 uses some of Java's native methods available via the `Packages` object. In this example, a user can type some information in a form text box, and then click the Print button to print his text to the Java Console. The `onClick` event is used to pass the text information to a JavaScript function, where it then writes the information. The result of running this example in a browser is shown in Figure 4.6.

NOTE

For this to work in Netscape 6, you must have selected the option to install the Java runtime during your initial installation.

Listing 4.4 Accessing Java Methods Through JavaScript's Packages Object

```

<html>
<head>
<title>LiveConnect Example #1</title>
<script language="JavaScript1.2" type="text/javascript">
<!--

//This function takes the text entered in by the user and prints
//it to the Java Console.
function writeToConsole(inText){
    Packages.java.lang.System.out.println(inText);
}
// -->
</script>
</head>

<body>
<form>
    <input type="text" name="entered" width="10">
    <input type="button" value="Print"
        onClick='writeToConsole(form.entered.value)'/>
</form>
</body>
</html>

```

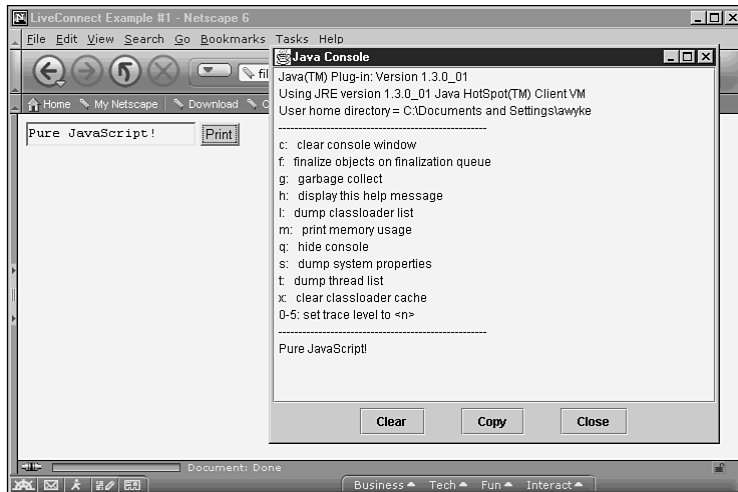


Figure 4.6

Using the *Packages* object to access the *System.out.println()* method in Java.

The second example is a little more complex. It takes text entered by the user and passes it to a method within an applet to change the text the applet is displaying. To help you get a full understanding of this example, this listing includes both the applet code and the JavaScript code needed to perform this task. Listing 4.5 shows the Java code necessary, whereas Listing 4.6 shows the JavaScript.

Listing 4.5 Java Code Used in the Example

```
// Imported classes
import java.applet.*;
import java.awt.Graphics;

public class PSLiveConnectExample extends Applet {

    // Initialize an instance of a public string variable
    public String sText;

    // Initial the string itself with a value
    public void init() {
        sText = new String("Pure JavaScript");
    }

    // Draw the string in the applet at this location
    public void paint(Graphics gArea) {
        gArea.drawString(sText, 20, 20);
    }

    // Method used to change the string when passed to the applet
    // from JavaScript.
    public void changeString(String aString) {
        sText = aString;
        repaint();
    }
}
```

Listing 4.6 HTML Used to Load the Applet and Change the Text Within It

```
<html>
<head>
<title>LiveConnect Example #2</title>
</head>

<body>
<center>
<table border="1">
  <tr>
    <td align="center">
      <applet
        code="PSLiveConnectExample.class">
```



```

        codebase="."
        name="TextChanger"
        width="150"
        height="25"
        mayscript>
    </applet>

    <form name="inputForm">
        <input type="button" value="Change Text"
            onclick="document.TextChanger.changeString(document.inputForm.
                ↪inputText.value)">
        <input type="text" size="20" name="inputText">
    </form>
</td>
</tr>
</table>
</center>
</body>
</html>

```

As you can see in Listing 4.6, the applet that displays the initial "Pure JavaScript" text is loaded. After the page has finished loading, the user can enter text into the text box and click the Change Text button to modify the text displayed by the applet. This button directly accesses the `changeString()` method in the applet to change the text from the default to that entered by the user. Figure 4.7 shows the result of changing this text.



Figure 4.7

Using JavaScript to change the text in a Java applet.

Form Processing

Before JavaScript was developed, most developers used server-side programs to send information from the Web server back to a user's browser after a form was submitted. When using a form, the user would enter his information and submit the form to the server. This usually was slow, due to transmission delay back and forth to the server and the fact that the Web server did most of the work with the form, such as parsing information and validation.

JavaScript helps speed up this process by interacting with the form data, similar to these server-side programs, before it is sent to the server. This saves the Web server work and reduces the amount of time spent transmitting information. Using JavaScript for form processing can be very useful to the developer and time-saving to the user.

NOTE

Form processing refers to the ability of Web developers to pass fields of information from the browser to be processed by a server. When this information is passed to the server, developers can use it as their application sees fit.

Many forms collect user information such as names, addresses, or interests. By collecting this information, developers open up a whole new level of possibility in their sites' functionality. They are no longer placing static information out on their site for the world to see; rather, it is a means of communication—communication between users and their sites.

Communication between users and sites is mostly done through forms. When a user fills out an online form and submits it, the information needs to be validated and processed. Without client-side scripting languages, these actions can only be done on the server-side. The validation and processing work can be broken out by using JavaScript, which is where the language's use of form processing really benefits developers.

Accurate processing of forms is dependent on the validity of the information given. It is very easy to input incorrect data into a form and then submit it. When this is done, you generally let the server handle any problems that might occur. These problems can be avoided if JavaScript is used on the client-side to validate the data before it is submitted. JavaScript can check the data content and format before transmitting the data, which, in turn, frees the server from encountering and dealing with these types of problems. The server's only job now is to process the information being sent.

FORM SUBMISSION 101: *GET VERSUS POST*

When creating a form, one of the attributes you must specify within the `<form>` tag is `method`. This attribute can be set to either `GET` or `POST`, both of which can be used as the type of submission for the form. However, their uses differ slightly.

`GET`, which is the default setting, passes the form values on to the URL. This is sent in a single transmission to the URL specified in the `action` attribute. The following

simple form can be used as an example:

```
<form action="http://www.purejavascript.com/cgi-bin/sampleform.cgi"
      method="get">
  <select name="state">
    <option value="NC">North Carolina</option>
    <option value="SC">South Carolina</option>
    <option value="CA">California</option>
  </select>
  <input type="button" value="Submit">
</form>
```

When the user selects one of the items from the pull-down menu and clicks the Submit button, the information will be passed on the end of the URL. If the user selected North Carolina, for instance, the passed information would look similar to the following:

```
http://www.purejavascript.com/cgi-bin/sampleform.cgi?state=NC
```

The **POST** method, on the other hand, works differently. When **POST** is used, the browser contacts the URL specified in the **action** property and sends the data in the body of the HTTP request. For those of you familiar with the HTTP standard, you know that information is passed back to the browser in two main parts: the **head** and the **body**. When a browser sends a **POST** form, it performs a task similar to the server in that it sends the information in the body of the request and not on the URL.

There are different cases for using the two methods. If you are sending large amounts of form information to the server, the **POST** method should be used. Normally a server will have problems handling large amounts of data using the **GET** method. The **POST** method is also useful for keeping your data from being easily seen because the form information is sent in a data block.

Keep in mind that JavaScript is not a complete substitute for server-side form processing programs. It should be used more as a complement to these programs. By combining the use of JavaScript and server-side programs, a developer can create a very effective and efficient means of processing forms.

Before programmers can use JavaScript with forms, they need to know how to get and parse forms for information. The next section takes a look at methods of extracting information from forms using JavaScript.

The Forms Array

Another JavaScript built-in object used to access components within a page is the Form array. The Form array, which is a property of the document object, is an array containing a list of the forms within a given document. Because it is an array, each form in the document can be referenced by the index number that represents its position in the document.

For instance, if you had a document containing three forms and you wanted to access a property in the second form, you would reference it with the following syntax:

```
document.forms[1].property
```

In this example, the *document* object refers to the document in which the form is located, and the *property* element refers to the property you are trying to access.

Listing 4.7 is an HTML page that contains two forms. In addition to the forms on this page, two JavaScript functions are included in the <head> portion of the page that show different methods of accessing forms and perform some input validation. Each of the forms contains a text area for the user to insert text, as well as a Submit button. When the user attempts to submit the form, an onClick event is fired and the contents of the form are passed to a JavaScript function for processing.

Listing 4.7 Using the *Forms* Array to Access Multiple Forms on a Page

```
<html>
<head>
<title>Forms Array Example</title>
<script language="JavaScript" type="text/javascript">
<!--

// Function verifyZip checks for a valid ZIP Code. If user enters invalid
// ZIP Code, then an alert box is used to inform the user. Function
// takes the length as an input parameter.
function verifyZip(length){

    // Create variable and store the form value for the ZIP input.
    var zipEntry = document.forms[0].zip.value;

    // Parse the input for an integer number using 10 as the radix.
    var zipNum = parseInt(zipEntry, 10);

    // Check to see that the length is 5.
    if (document.forms[0].zip.value.length == length){
        // Verify that the ZIP is a number.
        if(zipNum != 0 && isNaN(zipNum) == false){
            alert(zipEntry + " is a valid zip code");
        } else {
            // Inform the user if the ZIP is not valid.
            alert("Invalid Zip Code Entered. Please Re-enter");
        }
    } else {
        alert("Invalid Zip Code Entered. Please Re-enter");
    }
}

// Function myName displays the name the user entered.
```

```

function myName(){
    // Get the form value of the name entered.
    var name = document.form2.name.value;
    alert("You Entered: " + name);
}

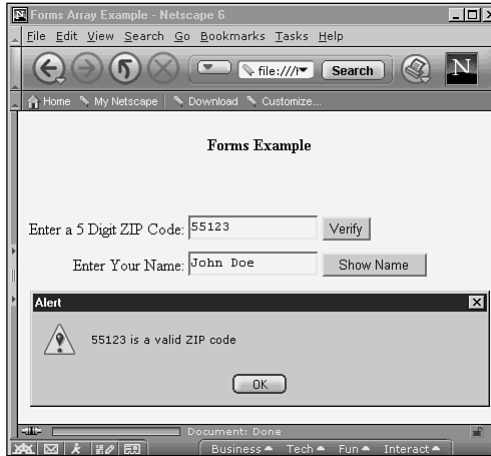
// -->
</script>
</head>
<body>
<p>
<center><b>Forms Example</b></center>
<br><br><br>
<table>
<form name="form1" method="post">
    <tr>
        <td align="right">Enter a 5 Digit Zip Code:</td>
        <td align="left">
            <input type="text" name="zip" size="15">
            <input type="button" name="button1"
                value="Verify" onclick="verifyZip(5)">
        </td>
    </tr>
</form>
<form name="form2" action="" method="post">
    <tr>
        <td align="right">Enter Your Name:</td>
        <td align="left">
            <input type="text" name="name" size="15">
            <input type="button" name="button2" value="Show Name"
                onclick="myName(this.form)">
        </td>
    </tr>
</form>
</table>
</body>
</html>

```

In the first form, validation is performed on the input. The user is asked to enter a five-digit ZIP Code. Once entered, the user can push the verify button, which calls the `verifyZip` function. This function uses the forms array index number to access the user input. Figure 4.8 shows this example of accessing the forms array.

NOTE

Remember that form indexes are zero indexed and are stored in the array in sequential order. The first form in the document is `forms[0]`, the second form is `forms[1]`, the third form is `forms[2]`, and so on.

**Figure 4.8**

The output result for processing the first form.

When a script has the input, it can perform checks to make sure that it is the correct length and that a numeric value was entered. If the correct information has been entered, a message is displayed informing the user the input was valid. If incorrect information was entered, an error message is displayed informing the user of an invalid entry.

The second form takes a name as input and passes it to the `showName` function, which displays the name in a window when the Show Name button is chosen. Notice that the function accesses the form information differently than the first function. Instead of using the forms array, it references the form by its name value.

In this example, the variable `name` is set using the `name` attribute of the `<form>` tag instead of the index number of the Forms array. Either method of accessing a form element is valid. However, if a document contains many forms, it might be easier to use the form name instead of counting the Forms array index numbers. Accessing a form through the name attribute also tends to make the script easier to understand.

This example demonstrates some uses of combining forms and JavaScript. Another useful function of JavaScript and forms is information manipulation. For instance, if the form processing program on the server-side only processes phone numbers without the hyphen, developers can program their scripts to strip all hyphens out of phone numbers before submission.

Accessing and Writing Information

Just as the document object contains a forms array, the form object contains an `elements` array. The `elements` array works similar to the forms array in that items are indexed in the order they appear, and you can access a specific element by its name

instead of index number. Listing 4.8 demonstrates the two different ways of accessing form elements.

Listing 4.8 Using Different Methods of Accessing Form Elements

```
<html>
<head>
<title>Example of accessing form elements</title>

<script language="JavaScript" type="text/javascript">
<!--

// Function displays the car information entered.
function showCar(){

    // Access the car information by using the elements array.
    var car = document.pref.elements[0].value;
    alert("Your favorite car is: " + car);
}

// Function displays the color information entered.
function showColor(){

    // Access the color value directly by form name.
    var color = document.pref.color.value;
    alert("Your favorite color is: " + color);
}

// -->
</script>
</head>
<body>

<form name="pref" method="post">
Enter the name of your favorite car:
    <input type="text" name="car" size="25">
    <input type="button" name="carButton" value="Show Car"
        onclick="showCar(this.form)">
<br>
Enter your favorite color:
    <input type="text" name="color" size="15">
    <input type="button" name="colorButton" value="Show Color"
        onclick="showColor(this.form)">
</form>

</body>
</html>
```

The information entered into each text box becomes an element of the form. Using the `elements` array, a script can access each individual element of the array. As you can

see in the `showCar` function, the car element is referred to by `document.pref.elements[0].value`. This is accessing the first element of the `elements` array in this specific form. The button would be the second element and would be referenced by `document.pref.elements[1].value`, and so on for other elements throughout the form.

The `showColor` function references the color object differently: by using the color object name with `document.pref.color.value`. If you were to use the `elements` array, this value would be referenced by `document.pref.elements[2].value`. Referencing by element name is much safer and easier to keep track of, and it can prevent future problems. For example, suppose that you have a Web page containing two frames, A and B, respectively. Frame A contains a JavaScript program that references elements within frame B using the `elements` array. If the Web page in frame B were to change but still contain form elements, your JavaScript program in frame A would be accessing incorrect form elements. Another reason for using element names is that if new form elements are inserted in the future, all the element numbering would not have to be changed.



TIP

Referencing by the `elements` array can be useful if you want to keep your program generic, but if you want better maintainability and ease of use, referencing by name is the preferred choice.

In addition to accessing form elements, you can also write or modify form information. This can be a very useful technique for automatic form correction. Just about any object that carries information within a form can be modified.

Take a look at a simple example of writing information to forms. Listing 4.9 contains a form with one text box with a question and one Submit button. The user is asked to enter the answer to the question. If the answer is correct, an information box appears indicating so. If it is wrong, an alert box appears informing the user that he entered the wrong answer, and the correct answer is automatically written to the text box.

Listing 4.9 Writing Information to JavaScript Forms

```
<html>
<head>
<title>Example of Writing Form Elements</title>

<script language="JavaScript" type="text/javascript">
<!--

// Function checks to see if the text submitted is the
// correct answer.
function checkText(){

    // Perform an equality check to see if the input is correct.
    if( document.FormExample3.textbox.value == "Bugs Bunny"){
```



```

        alert("You are correct!");
    } else {

        // If the original input was incorrect, output the
        // correct information and inform the user.
        document.FormExample3.textbox.value = "Bugs Bunny";
        alert("That is incorrect. The correct answer is now in the text box.");
    }
}

// -->
</script>
</head>
<body>

<form name="FormExample3" method="post">
What Looney Tunes Character is gray and has long ears?
    <input type="text" name="textbox" size="25">
    <input type="button" name="Bugs" value="Submit"
    ↪ onclick="checkText(this.form)">
</form>
</body>
</html>

```

Building on the foundation of the code in Listing 4.9, it's possible to create a JavaScript method that creates a customized pull-down menu based on specific user input.

Form Example

Listing 4.10 presents a sample Web page that allows a user to enter his personal information to submit. After all the information is entered and the user clicks the Submit button, all the form data is validated on the client-side before passing the data to the server for further processing.

Listing 4.10 Example Using a Form for Client-Side Validation

```

<html>
<head>
<title>Form Validation Example</title>
<script language="JavaScript" type="text/javascript">
<!--

// Function checks to see that the personal
// information entered is valid.
function validatePersonalInfo(){

    // Declare variables to hold input values.
    var _first = document.info.fname.value;
    var _last = document.info.lname.value;
    var _street = document.info.street.value;

```

Listing 4.10 Continued

```
var _city = document.info.city.value;
var _zip = document.info.zip.value;
var _phone = document.info.phone.value;
var _email = document.info.email.value;

// Verify that the all input fields are filled in.
if(_first.toString() == ""){alert("Please enter a first name.");}
if(_last.toString() == ""){alert("Please enter a last name.");}
if(_street.toString() == ""){alert("Please enter your street name.");}
if(_city.toString() == ""){alert("Please enter your city.");}
if(_zip.toString() == ""){alert("Please enter your ZIP.");}
if(_phone.toString() == ""){alert("Please enter your phone number.");}
if(_email.toString() == ""){alert("Please enter your email.");}

else{
    // Check that the ZIP and phone numbers are valid inputs.
    var checkZip = checkNum(5);
    var phoneInput = document.info.phone.value;

    // Initialize variables.
    var validPhone = false;
    var validZip = false;

    if(checkZip == true){
        validZip = true;
    }
    else{
        alert("Invalid ZIP Code: " + validZip);
    }

    // If the phone number is not valid, then inform user.
    if(!checkPhone(phoneInput)){
        alert("Phone number is invalid." + validPhone);
    }
    else{
        validPhone = true;
    }

    if(validZip && validPhone){
        alert("Your form has been verified");
    }
}
}
```



```
// Strips hyphens out of phone number and verifies that
// phone number is valid. Any phone number in the format
// xxxxxxxxxx, xxx-xxx-xxxx, or (xxx)xxx-xxxx will be valid.
```

```

function checkPhone(str){
    var regexp = /^(\d{10}|\d{3}-\d{3}-\d{4}|\(\d{3}\)\d{3}-\d{4})$/;
    return regexp.test(str);
}

// Function checks that the ZIP Code is valid.
function checkNum(length){
    var zipEntry = document.info.zip.value;
    var zipNum = parseInt(zipEntry, 10);

    if (document.info.zip.value.length == length){
        if(zipNum != 0 && isNaN(zipNum) == false){
            // Valid ZIP Code
            return true;
        }
        else {
            // Invalid ZIP Code
            return false;
        }
    }
    else {
        //Too Many digits- Invalid
        return false;
    }
}

// -->
</script>
</head>
<body>
<p>
<center><b>Form Validation Example</b></center>
<p>This page demonstrates how JavaScript can do form validation on a Web page.
Using JavaScript to do validation on the client-side can greatly reduce
processing time by reducing the chance of submitting incorrect forms.
<hr>
<br>
<b>Personal Information:</b>
<form name="info" method="post">
<table>
  <tr>
    <td align="left">first name:</td>
    <td align="left">
      <input type="text" name="fname" size="15">
      last name:
      <input type="text" name="lname" size="20">
    </td>
  </tr>
</table>
<br>

```

Listing 4.10 Continued

```
<tr>
  <td align="left">Street:</td>
  <td align="left">
    <input type="text" name="street" size="30">
  </td>
</tr>
<br>
<tr>
  <td align="left">City:</td>
  <td align="left">
    <input type="text" name="city" size="15">
    State:
    <select name="state">
      <option value="AL">AL</option>
      <option value="AK">AK</option>
      <option value="AZ">AZ</option>
      <option value="AR">AR</option>
      <option value="CA">CA</option>
      <option value="CO">CO</option>
      <option value="CT">CT</option>
      <option value="DE">DE</option>
      <option value="FL">FL</option>
      <option value="GA">GA</option>
      <option value="HI">HI</option>
      <option value="ID">ID</option>
      <option value="IL">IL</option>
      <option value="IN">IN</option>
      <option value="IA">IA</option>
      <option value="KS">KS</option>
      <option value="KY">KY</option>
      <option value="LA">LA</option>
      <option value="ME">ME</option>
      <option value="MD">MD</option>
      <option value="MA">MA</option>
      <option value="MI">MI</option>
      <option value="MN">MN</option>
    </select>

    ZIP:
    <input type="text" name="zip" size="7">
  </td>
</tr>
<br>
<tr>
  <td align="left">Phone (w/area code):</td>
  <td align="left">
    <input type="text" name="phone" size="20">
  </td>
</tr>
```

```

<br>
<tr>
  <td align="left">Email:</td>
  <td align="left">
    <input type="text" name="email" size="20">
  </td>
</tr>
<br>
</table>
<center>
  <input type="button" value="Submit" onclick="validatePersonalInfo()">
</center>
</form>
</body>
</html>

```

Figure 4.9 shows how the personal information form looks.

The screenshot shows a web browser window titled "Form Validation Example - Microsoft Internet Explorer". The browser's address bar shows a local intranet path. The page content includes a title "Form Validation Example" and a paragraph explaining that the page demonstrates how JavaScript can perform form validation on a web page to reduce processing time by preventing incorrect submissions. Below this is a section titled "Personal Information:" containing several input fields: "First name:" and "Last name:" (text boxes), "Street:" (text box), "City:" (text box), "State:" (a dropdown menu currently showing "AL"), "ZIP:" (text box), "Phone (w/area code):" (text box), and "Email:" (text box). A "Submit" button is located at the bottom right of the form area. The browser's status bar at the bottom indicates "Done" and "Local intranet".

Figure 4.9

Form example using client-side validation.

The form begins by validating that all the input fields have been filled. If any of the fields are left empty, an alert box will appear indicating that the field requires input. Two main inputs require special validation: the ZIP Code and the phone number.

The first function called is `ValidatePersonalInfo`. This main function checks to see that all the user entries are valid. If so, an alert box is returned indicating that the form has been validated. Before the form can be completely validated, the ZIP Code and

phone number must be checked. The ZIP Code verification is performed by the sub-function called `checkNum`. This function takes one parameter—the string length. For the ZIP Code, we are restricting the string length to five digits. If this is found to be true, the ZIP Code is validated.

The second sub-function called is `checkPhone`. The `checkPhone` function takes the input for the phone number and checks it against three different standard phone number formats. If any are found to be valid, the function returns `true`. After both checks are performed, the form is determined to contain valid information.

Cookie Handling

A *cookie* is a small bit of organized information that is stored by a browser in a text file on the user's computer. Cookies are typically used to store information pertinent to a specific site that is currently used and can be reused in the future.

The location of cookie text file differs according to the browser being used. In Navigator, all cookies are stored in a file named `cookies.txt`. In Internet Explorer, each cookie is stored as its own individual text file in the Cookies folder, which is located by default in the `\Windows` or `\Winnt` folder.

There are limitations to cookies. The size of a cookie is limited to 4KB. Also, Navigator browsers only allow for 300 total cookies to be stored on the user's computer. This keeps the size of the `cookies.txt` file or Cookies folder to a 1200KB maximum. In addition, each Web server is only allowed to store a total of 20 cookies. If the 300 total cookies or 20 cookies per Web server limit is exceeded, the least recently used cookie is deleted to accommodate any additional cookies.

Browsers let the user control how cookies are used. In Navigator, under Edit, Preferences, there is an Advanced category. This category gives you the option to accept all cookies, accept cookies that get sent back to the server, or disable cookies. There is also an option to warn you before accepting a cookie. Enabling this option will force the browser to notify you when a cookie requests to be set and inform you of what the cookie contains. Figure 4.10 shows a picture of what a cookie warning would look like. Similarly, Internet Explorer controls the use of cookies through its Security tab on its Internet Options dialog box (available under the Tools menu for version 5.5 on Windows).

NOTE

When a cookie is first set, it is stored in the browser's memory. It isn't until the browser is exited that the cookie gets written to the file.

The cookie object is part of the Document object. Cookies can be created, set, and modified by setting the appropriate values of the cookie property. A cookie has four name attributes: `expires`, `path`, `domain`, and `secure`.

**Figure 4.10**

Cookie warning example.

By default, a cookie lasts only during the current browsing session. When the browsing session is over, the cookie is destroyed. For a cookie to last beyond the current browsing session, the expires attribute must be set. This attribute specifies the life of a cookie. The value of this attribute can be set to any valid date string. If the expires attribute is set, the cookie is written to the cookie text file after the current browsing session is over. If no expires attribute is set, the cookie will expire when the user's browsing session ends.

The path attribute specifies the domain associated with the cookie. The level of association begins at the specified path and goes down into any subfolders. For example, suppose that `http://www.purejavascript.com/examples/cookie.html` was setting a cookie and wanted the cookie to be shared across Web pages on the `pure-javascript.com` domain. To do this, the cookie path attribute needs to be set to `"/`". This allows the cookie to be accessed from any page on the `www.purejavascript.com` Web server. If the path was set to `"/examples"`, the cookie would only be valid to pages in the examples folder and its subfolders.

NOTE

If the `secure` attribute is specified, the cookie will only be transmitted over a secure channel (HTTPS). If `secure` is not specified, the cookie can be transmitted over any communications channel.

Understanding the cookie attributes is the first step in being able to read and write cookies. Now that you do understand this, look at how to read cookies.

Reading Cookies

As stated before, cookies are part of the document object. The first thing you have to do is to read the cookie property. This can be done with the following statement:

```
var cookieName = document.cookie;
```

This statement creates the variable `cookieName` and assigns it the `cookie` property of the document object. When you have accessed this property, you must extract the various attributes of the cookie. The `cookie` property itself returns a string containing all the cookies pertaining to the current document.

Cookies are interpreted in name/value pairs. The string returned from the `cookie` property contains the list of cookie name/value pairs. To read an individual attribute, you must parse through its string. You can use built-in JavaScript string methods to accomplish this.

Look at the example in Listing 4.11 to see how to read a cookie. When calling this function in your code, you simply need to pass the name of the cookie you are looking for.

Listing 4.11 *Getting a Cookie Value*

```
<script language="JavaScript" type="text/javascript">
<!--
```

```
// Declare variable to hold all the cookies contained
// in the document.
var cookies = document.cookie;
```

```
// Function that gets a cookie's value by searching
// for the name of the cookie.
function readCookie(name) {
```

```
    // Declare variable to set the "name=" value.
    var start = cookies.indexOf(name + "=");
```

```
    // Get the index if the cookie name is found.
    if (start == -1){
        alert("Cookie not found");
    }
```

```
    // Get the first character of the cookie.
    start = cookies.indexOf("=", start) + 1;
```

```
    // Read to the end of the cookie.
    var end = cookies.indexOf(";", start);
```

```
    if (end == -1){
        end = cookies.length;
```



```

}

// Get the cookie value, reversing the escaped format by
// using the unescape method.
var value = unescape(cookies.substring(start, end));

if(value == null){
    alert("No cookie found");
}
else{
    alert("Cookie value is: " + value);
}
}
// -->
</script>

```

Writing Cookies

Cookie values can be created, modified, and set. Remember that cookies are read as name/value pairs. Therefore, when you write a cookie, its attributes must be written as a name/value pair.

To set a cookie value, you first create a cookie and assign it a name. Then you must set each individual attribute for the new cookie. Keep in mind that when you set the expires attribute, you must have a valid date string. It is best to use the JavaScript Date methods to create a date string, which can be assigned. Additional cookie attributes can be set similarly. The following is an example of how this is done:

```
document.cookie = "name=" + form.cookie.value + "; expires=" + month;
```

If you want to modify a specific value in the cookie, just set its attributes again. You can delete a cookie's value by modifying that attribute.

Windows in Action

A window object is used in all your JavaScript programs, whether you specify it or not. The window object is one level higher than the document object in the JavaScript object hierarchy and provides a developer with the ability to manipulate the current window as well as create new window instances.

The ability to create new windows allows developers to add functionality to their sites. Some sites have advertisers create new windows to show their advertisement banners, whereas others, such as ESPN (shown in Figure 4.11), use this as a method to play games in windows while their users continue to browse.

The Parent/Child Relationship

Windows use what is described as a parent/child relationship when they are initiated. This means that if window B is created from within window A, window B is considered a child of window A, which is its parent. Any actions performed on the child by its parent must be referenced through its parent. Look at the following code:

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--
// Function creates a new window called myChild
// with the index.html file in it.
function create(){
    open("index.html", "myChild");
}
// -->
</script>
</head>
<body>
<form name="form1">
    <input type="button" value="Create Child" onclick='create()'/>
</form>
</body>
</html>

```



Figure 4.11

Pop-up window for a game.

The function `create` simply creates a new window that would be considered a child of the original window. The name of the child is specified in the second parameter of the `window.open` function, which is `"myChild"`. Any values in the child window are referenced using the name of the parent window. If the child window has a child of its own,

this would be considered a grandchild of the parent. For a grandparent to perform actions on its grandchild, it must first reference its immediate parent and then its grandparent. Suppose that the grandchild is named Jane, its parent is named Bob, and the grandparent is named Joe. For Jane to be manipulated by Joe's window, it must reference the value by the following:

```
joe.bob.jane.document.form1.textBox.value
```

The parent has total control of its immediate child window and any subsequent child (grandchild) windows. If the parent window is destroyed, all control for the remaining child windows is lost.

A child window can find out who its parent is by using the `opener` method. The `opener` method will tell the child from which window it was created. You use the `opener` method as shown in the following:

```
var name = window.opener.document.name;
```

This will return the name of the parent document to the `name` variable.

You can use multiple instances of `opener` if you have several levels of windows. If a grandchild wanted to reference its grandparent, it could do so by simply adding another level:

```
window.opener.opener.document.form1.textbox.value
```

For a window to refer to its own properties, you can use the `window self` property. Suppose that you wanted to set a `textbox` value within a form document in a window. You could refer to the value as shown in the following:

```
self.document.formName.textbox.value = "JavaScript is cool";
```

Creating Windows

JavaScript provides a built-in method to create new instances of windows. The `open` method can be used to create any primary or secondary window. The following is an example of how `open` can be used.

```
window.open("web.html", "newWin", "resizeable,menubar,toolbar");
```

This creates a window that is resizable, contains a menu bar, and has a toolbar. The `"web.html"` is the file that will open in the newly created window. `"newWin"` represents the name of the window object. Many different features can be added to a window. This example only shows a few. Other features include, `outerHeight`, `outerWidth`, `innerHeight`, `innerWidth`, `alwaysRaised`, `alwaysLowered`, `location`, `screenY`, and `screenX`. The features are specified in the parameters of the `open` function.

NOTE

Be careful not to include any spaces between window features; this will cause errors to occur in your window creation. Refer to the `windows` object in Chapter 8, "Client-Side," for more details on other windows properties.

After you create a new window, you can manipulate it. JavaScript also provides you with the ability to interact between multiple windows. As long as a window property is referenced correctly, you can modify objects in other windows. Remember the parent-child relationship when referencing window objects.

Limitations

In older versions of JavaScript, windows could only be referenced from the top down. This meant that a parent window would know the properties of its child windows, but not vice versa. In JavaScript version 1.2 and later, this problem is resolved. As mentioned earlier, using the `opener` property will resolve any child/parent referencing conflicts.

Window Example

The next example shows how you can send information between two windows. It begins with the parent window. Figure 4.12 shows both the parent and child windows open with the parent window on the top.

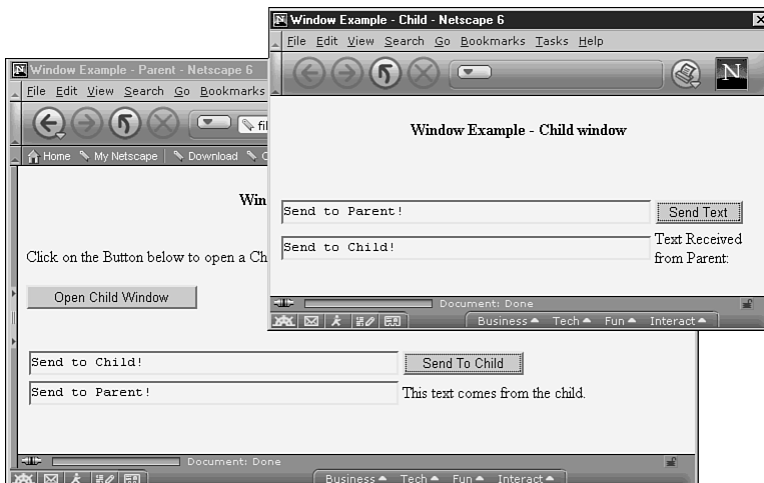


Figure 4.12

Window example showing how information can be sent between windows.

Look at the code for the parent window in Listing 4.12.

Listing 4.12 Parent Code for Window Example

```

<html>
<head>
<title>Window Example - Parent</title>
<script language="JavaScript" type="text/javascript">
<!--

// Function creates and opens a new child window.
function openWin(){
    // Open a new window named "ChildWin".
    childWin=open("child.html", "ChildWin", "toolbar,scrollbars,menubar,status,
        width=500,height=200");
}

// Function sends a value from the parent window to the
// child window.
function sendToChild(){
    childWin.document.childForm.childText.value =
        document.parentForm.parenttext.value;
}

// -->
</script>
</head>

<body>
<p>
<center>
    <b>Window Example - Parent Window</b>
    <br><br><br>
</center>
<form name="parentForm">
    Click on the Button below to open a Child window:
    <br><br>
    <input type="button" name="submit" value="Open Child Window"
        onclick="openWin()">
    <br><br><br>
    <table>
        <tr>
            <td align="left">
                <input type="text" name="parenttext" size="45">
            </td>
            <td align="left">
                <input type="button" name="submit" value="Send To Child"
                    onclick="sendToChild()">
            </td>
        </tr>
    </table>
    <tr>

```

Listing 4.12 Continued

```
<td align="left">
    <input type="text" name="received" size="45">
</td>
<td align="left">
    This text comes from the child.
</td>
</tr>
</table>
</form></body>
</html>
```

This parent window has a button, which, when clicked, opens a child window. It also has two text fields, one for sending text to the child and one for receiving text from the child. The user will first click the button to create a new window. This is the child window. When the user enters text into the first text box and clicks the Send Text button, the input is sent to the child and appears in its appropriate text box. Take a look now at what the child window looks like in Listing 4.13.

Listing 4.13 Child Window Code for Window Example

```
<html>
<head>
<title>Window Example - Child</title>
<script language="JavaScript" type="text/javascript">
<!--

// Function sends a value from the child window to the parent window.

function sendToParent(){
    window.opener.document.parentForm.received.value =
        document.childForm.sendText.value;
}

// -->
</script>
</head>

<body>
<p>
<center>
    <b>Window Example - Child window</b>
    <br><br><br>
</center>
<form name="childForm">
    <table>
        <tr>
            <td align="left">
```

```

        <input type="text" name="sendText" size="45">
    </td>
    <td align="left">
        <input type="button" name="submit" value="Send Text"
            onclick="sendToParent()">
    </td>
</tr>
<br>
<tr>
    <td align="left">
        <input type="text" name="childText" size="45">
    </td>
    <td align="left">
        Text Received from Parent:
    </td>
</tr>
</table>
</form>
</body>
</html>

```

The child window is similar to the parent except that there is no option to open another window. The child window has one text field for sending text to the parent window and one text field for receiving text from the parent. In the JavaScript code, you can see the use of the `opener` property to reference its parent. Remember, you cannot reference windows directly upward. The parent can call a child by name, but a child cannot call the parent by its name.

Manipulating windows provides you with the ability to greatly expand your site's functionality. Many window properties can be specified or modified to your individual needs. Do be careful on how you implement these windows, however, because often times too many (or any for that matter) pop-up windows can be annoying. Be sure that you really need it.

Dynamic Documents

As Web sites became more advanced, the need for dynamic content grew. In the beginning, the only type of presentation technologies available was plain HTML. It allowed various type of formatting so content could be displayed in a pleasant manner, but soon more was needed—enter scripting languages. The combination of JavaScript and HTML was a big step, and that coupled with support for CSS meant that a developer could create static content with HTML and then have control over it with JavaScript and CSS.

The Web has come a long way in a very short period. The first generation was called Dynamic HTML or DHTML for short. As we just mentioned, DHTML wasn't one specific technology, but a core set of three technologies working together to provide a toolset for developers to create dynamic content. But things have moved on even more since then.

In today's newest browsers we have the *Document Object Model (DOM)* at our disposal. This is an attempt to standardize the method in which you access elements within a given document, such as an HTML or XHTML document. This is definitely the way of the future, but it does not remove the fact that some of the older browsers are still around.

Within this section of the chapter, we are going to take a look at both the old (DHTML) and new (scripting the DOM) ways of doing things. We will also show you how to build cross-browser scripts that will work in all version 4+ browsers. With the rate of adoption of the DOM, this will most likely be the last edition of this book that covers the old way, but for now it is definitely still needed.

The Old Way

Netscape initially tackled three-dimensional positioning with the `<layer>` and `<ilayer>` tags in Navigator 4. Internet Explorer 4, on the other hand, implemented three-dimensional positioning through their *DHTML Object Model* implementation by extending the functionality of the existing HTML tags, such as `<div>`.

With the release of these version 4 browsers, developers had two ways of creating layers within documents, but they were browser-specific. Cascading Style Sheets, which is also needed for creating dynamic documents, has been implemented in both browsers. CSS provides the base positioning. There are some differences in implementation, in both level and interpretation, so this area is still in a state of change, but things are getting much better.

Navigator 4 Layers

The concept of layers was introduced in Navigator 4, but not only in the HTML, but also through the new Layers object in JavaScript 1.2. Similar to many things at the time, these layers were only supported in Navigator 4—not Internet Explorer. They did have some usefulness, however, by providing the ability to control the stacking and movement of elements within a single window.

On the HTML side, one of the new tags created was the `<layer>` tag. This tag allowed you use JavaScript for layer manipulation. The following shows an example of using this tag.

```
<layer id="myLayer" width="220" height="100" bgcolor="red" top="150"
      left="100" visibility="show">
  This is a Navigator 4 tag
</layer>
```

The positioning of layers could be either absolute or relative, as with most elements. With *absolute* positioning, you can specify the position within its containing layer. *Relative* positioning is when the object appears in its natural location within the flow of the document.

The `<layer>` tag used absolute positioning, and contained properties that let a developer specify the exact position in the document. Now the second tag they introduced

allowed for relative positioning. The `<ilayer>` tag was considered an inflow or inline layer.

The HTML tags were only half of what was added by Netscape for this functionality. In addition to this, they also added the `Layer` object and the `document.layers` array to JavaScript for accessing and manipulating these layers. This object contained its own methods and properties for performing these tasks.

Even though it is not defined in the HTML 4 Recommendation, Navigator 4 required that you have the `name` attribute set in the tags, be it `<layer>`, `<ilayer>`, or `<div>` (which was used to define groups of tags) for this to work. The syntax is rather simple for accessing layers in this manner. For instance, if you are wanting to access a layer called `layer1`, you could accomplish this through the following syntax:

```
document.layers['layer1']
```

Because these types of layers represent the old method of performing document dynamics, we are not going to go into any more detail here. We will, however, include a cross-browser example later in the chapter in the section called “Dynamic Documents” that will help show how dynamic documents in Navigator 4 works. But moving forward, you should pay close attention to the new method, which is what has been implemented in Netscape 6.

NOTE

Want some more information on Navigator’s `Layer` object? Check out its entry in Chapter 8.

Internet Explorer 4’s DHTML Object Model

Internet Explorer 4 took a broader approach to DHTML by allowing developers the ability not only to access layers, which they defined primarily using `<div>` tags, but also to access almost all tags within a given document. On the layer front, the `<div>` approach was the way to go because it was a tag that was part of the HTML Recommendation. The tag is essentially used to define data blocks within a document—blocks that can later be scripted, styled, and manipulated.

In addition to this tag, there was also the `<span>` tag. Whereas the `<div>` tag provided a way to organize numerous tags together as a single group, `<span>` was used to group a set of characters inline, within the body of another tag.

Similar to Navigator, Internet Explorer needed a method to access these elements and manipulate them. Microsoft did not implement the `Layer` object, but rather the `document.all` collection. This collection provided the means by which you could access elements within your document. However, unlike the `document.layers` method, it did not rely on the non-standard `name` attribute of the `<div>` tag, but rather on the standardized `id` attribute instead. Using the same `layer1` as our example, you would access this layer in Internet Explorer 4 or 5 using the following syntax:

```
document.all['datablock1']
```

Even though both the Netscape and Internet Explorer methods differ, both have been deprecated in favor of the DOM approach.

The New Way

The Document Object Model (DOM) is a very broad topic, and covers all aspects of representing documents within an object model designed for accessibility and flexibility. In this section, we will discuss the basics of the DOM, its structure, and how it can be used in relation to JavaScript. This is meant to be an overview of the DOM.

NOTE

If you really want to learn more about the DOM and all its details, take a look at the W3C DOM Level 1 and Level 2 Recommendation and the new work involved around the future Level 3.

What Is the DOM?

The DOM refers to the Document Object Model, which provides access to all elements and their attributes within a document through a hierarchical structure. The DOM API can be used to create, delete, and modify elements and content. Although the DOM was designed to work with any programming language, it is most commonly associated with XML and HTML. The primary focus here will be within the scope of the JavaScript binding that it defines.

The DOM Level 1 is supported starting in Netscape 6, Internet Explorer 5 and Opera 5, whereas parts of Level 2 are also supported in these browsers. As usual, each browser will have its own quirks on how parts of the DOM work. Earlier versions of these browsers also support various aspects of the DOM, but not completely.

Manipulating the DOM

Before getting into discussions on how to use the DOM, we need to set up a little vocabulary. When talking about the DOM, three terms are often used. They are node, element, and attribute. Each document contains a bunch of *nodes*, which represents either a piece of text or a tag and its attributes. A tag (that is, `<img>`) is also referred to as an *element* in terms of the DOM. Each element might or might not have an *attribute*. For example in the following HTML code, we have the `<img>` element containing attributes of `src` and `name`:

```
<html>
<body>

</body>
</html>
```

The DOM allows a program to access all the elements in a document through a “tree” like object structure. The structure begins with a root object; typically in an HTML file, this would be the `<html>` tag. Within the root object, additional objects are contained in a parent-child type relationship. A child would be considered as any node nested

within another node. In the previous example, the `<html>` node would be a parent to the `<body>` node. The `<img>` node would be considered a child to the body node, and so on.

Nodes can be accessed directly by name (if they have a name attribute) by using built-in DOM methods or through parent-child relationship properties. To access a node by name, use the `getElementById()` method of the Document object. If using the parent-child assessors, the `Document.parentNode[]`, `Document.childNodes[]`, and `Document.getSiblings[]` arrays will be the most useful.

Elements and attributes can be added, modified, or deleted through the DOM. To create a new element, the `createElement()` method is used. Looking at the following example, we see how to create a new anchor element and add attributes to it:

```
var myNewLink = document.createElement('a');
myNewLink.setAttribute('href', 'home.html');
myNewLink.setAttribute('name', 'Homepage');
```

There are many built-in methods and properties for manipulating the DOM. To see a list of all the properties, refer to the W3C DOM Recommendation available at <http://www.w3.org/DOM> or the DOM browser support information for Netscape, Internet Explorer, or Opera browsers.

DOM Cascading Style Sheets

The DOM can also be used with CSS, which is often used to create Web pages requiring positioning, formatting, or dynamic content. It allows you to define constructs on how your content is presented, and the DOM can be used to expose these constructs to an object model. Once exposed, it allows a JavaScript program to create, modify, or remove CSS properties.

All CSS items should be available within the document interface. In addition, the `CSSStyleSheet` interface can also be used to access the rules for a particular style sheet. To determine whether the DOM has support for CSS, the `hasFeature()` method can be called passing CSS for the first argument (feature) and the version (Level) number.

DOM Example

Now that we have discussed what the DOM is, some of its methods, and how to manipulate it, let's take a look at an example. Listing 4.14 demonstrates three different parts; the first of which is how to add text dynamically within a document. The second shows how to modify an HTML ordered list to an unordered list, and the third part shows how a link can be added to a document dynamically.

Listing 4.14 *Example of Different DOM Uses*

```
<html>
<head>
  <script "type="text/javascript" language="JavaScript">
    <!--
```

Listing 4.14 Continued

```
// adds a piece of text to a document
function addtext(text){
    if (document.createTextNode){
        var mytext=document.createTextNode(text);
        document.getElementById("example").appendChild(mytext);
    }
}

// changes an ordered list to an unordered list
function changeList(){
    var oldListItems = myList.innerHTML;
    var newNode = document.createElement("UL");
    myList.replaceNode(newNode);
    newNode.innerHTML = oldListItems;
}

// adds link to the document
function addLink(){
    var myLink = document.createElement('a');
    myLink.setAttribute('href', 'http://www.amazon.com');
    myLink.setAttribute('name', 'Amazon');
    text = document.createTextNode(' Amazon.com ');
    myLink.appendChild(text);
    document.getElementById('firstlink').appendChild(myLink);
}
// -->
</script>
</head>
<body>
<h3>This is an example of how to use the DOM to perform various tasks</h3>
<!-- First DOM Example -->
<p>
This example shows how to dynamically add new text into a document.
<div id="example" onClick="addtext('*This is new text* ')">
    <b>Click here</b> to add the new text.
</div>
<p>
<hr>
This example shows how to modify the ordered list to an unordered list.
<br>
( works in IE 5 )<br>

<!-- Second DOM Example -->
<form name="form1">
    <ol id=myList>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
```

```

    </ol>
    <input type="button" value="Change List Type" onclick="changeList()">
</form>
<p>

<!-- Third DOM Example -->
This example shows how to dynamically add a new link
<div id="firstlink" onClick="addLink()">
    <b>Click here</b> to add the new text.
</div>
<hr>

</body>
</html>

```

Cross Browser DHTML

So far, we have discussed the two ways of creating and handling dynamics in HTML documents: the old DHTML way, and the new DOM way. Earlier in the chapter, we talked about the old way and then showed the basic syntax of how it worked. Later, we discussed the newer DOM method and included an example. Now, we are going to show you a single piece of code that will show and hide a data block, or layer if you will, that works not only in the older version 4 browsers using the DHTML methods, but also in the newer version 5 and 6 browsers using the DOM approach.

As you will see in our example, we will use `<div>` tags to create our blocks, CSS to perform some basic styling, and finally JavaScript to determine the type of browser the user has and how to handle the scripting of the data blocks. In this example we will create a data block, or layer, and two form buttons. When one button is clicked the block will disappear. When the other one is clicked, it will reappear.

There are several aspects of this listing to review, starting with the simple aspects first. The HTML used is nothing more than a simple form with two buttons that call JavaScript functions on `onclick` events. Directly below this form is an even more simple `<div>` block that only contains the text "My block". Also notice that we use the `onload` event handler within the `<body>` tag to call a JavaScript function. This code looks similar to

```

<body onload="checkBrowser()">
    <form>
        <input type="button" value="Hide"
            onclick="changeState('datablock','hidden')">
        <input type="button" value="Show"
            onclick="changeState('datablock','visible')">
    </form>
    <div name="datablock" id="datablock">
        <p>My block</p>
    </div>
</body>

```

As for the function, its purpose is to set the proper variables that represent our scripting syntax. It looks like the following, where `ieVer` represents the major version number of any IE browser accessing the page. As for the "Gecko" reference, that is part of the user agent string within browsers built from Mozilla.org's Gecko rendering engine, such as Netscape 6.

```
function checkBrowser(){
    // IE 4
    if((ieVer < 5) && (ieVer > 0)){
        block = ".all";
        style = ".style";
        isDHTML = "true";

    // Navigator 4
    }else if(navigator.userAgent.indexOf("Nav") != -1){
        block = ".layers";
        style = "";
        isDHTML = "true";

    // IE 5+ and Netscape 6+
    }else if((navigator.userAgent.indexOf("Gecko") != -1) ||
        (ieVer >= 5)){
        isDOM = "true";
    }
}
```

Within Navigator, as we had mentioned, this is accomplished by including the property you want to access immediately after the `document.layers['layerName']` declaration. For Internet Explorer, however, this is accessed through the additional `style` collection, so your syntax will be something similar to `document.all['layerName'].style.property`.

These two methods might seem to pose a problem for writing your code because it appears we will have to write two different sets of code for each browser. Within the JavaScript language, however, there is a top-level `eval()` function that takes a string passed to it, and evaluates it as JavaScript code. This allows us, for instance, to dynamically build a piece of JavaScript code, such as our layer reference, and pass it to the `eval()` function to be executed.

This only takes care of the older DHTML method, so we also need to take what we learned earlier and use it to access the DOM for the newer browsers. We accomplish this by using the `getElementById()` method to return a pointer to the layer, and then using the `element.style.visibility` property to hide or show the layer. You will also notice in the function that we set an `isDHTML` and an `isDOM` variable, which we will use later.

The last function takes care of all this and is the one called by our form buttons—`changeState()`. This function takes two parameters—the first one is the name of the

layer it is supposed to effect, and the second is the visibility value that the layer should be changed to. Because we have to access and change the layer properties using different methods, our function will look similar to

```
function changeState(dblock, state){
    if(isDHTML == "true"){
        eval("document" + dblock + "[" + dblock + "]" + style +
            ".visibility = '" + state + "'");
    }else if(isDOM == "true"){
        var blockElement = document.getElementById(dblock);
        blockElement.style.visibility = state;
    }
}
```

At this point, we have gone over most of the code with the exception of some variable declarations and a single style sheet property setting. Listing 4.15 contains the complete example.

Listing 4.15 Manipulating Layers in All Supporting Browsers

```
<html>
<head>
    <style type="text/css">
        <!--
            #datablock{
                background-color: lightblue;
            }
        -->
    </style>
    <script type="text/javascript" language="JavaScript1.2">
        <!--
            // global variables for browser
            var block = new String();
            var style = new String();
            var isDHTML = new Boolean(false);
            var isDOM = new Boolean(false);

            // determine IE version
            var ieVer = parseInt(navigator.userAgent.charAt(parseInt(
                ↪navigator.userAgent.indexOf("MSIE")) + 5));

            // set appropriate variables depending on scripting method
            function checkBrowser(){
                if((ieVer < 5) && (ieVer > 0)){
                    block = ".all";
                    style = ".style";
                    isDHTML = "true";
                }else if(navigator.userAgent.indexOf("Nav") != -1){
                    block = ".layers";
                    style = "";
                }
            }
        </script>
    </head>
</html>
```

```
        isDHTML = "true";
    }else if((navigator.userAgent.indexOf("Gecko") != -1) ||
        (ieVer >= 5)){
        isDOM = "true";
    }
}

// Take the state passed in, and change it.
function changeState(dblock, state){
    if(isDHTML == "true"){
        eval("document" + block + "[" + dblock + "]" + style +
            ".visibility = '" + state + "'");
    }else if(isDOM == "true"){
        var blockElement = document.getElementById(dblock);
        blockElement.style.visibility = state;
    }
}
}
//-->
</script>
</head>
<body onload="checkBrowser()">
    <form>
        <input type="button" value="Hide"
            onclick="changeState('datablock','hidden')">
        <input type="button" value="Show"
            onclick="changeState('datablock','visible')">
    </form>
    <div name="datablock" id="datablock">
        <p>My block</p>
    </div>
</body>
</html>
```

XML-Based User Interface Language (XUL)

In this section, we will discuss the basics of a new language called the XML-based *User Interface Language (XUL)*. XUL works across many different platforms within the Gecko rendering engine and with various programming languages. We will attempt to cover XUL from a high level and provide some insight into how JavaScript can be used in conjunction with XUL.

What Is XUL?

XUL is an XML-based programming language used for defining objects and layouts in a user interface. XUL was created to provide a XML shorthand for describing the contents of windows and dialogs and is heavily used within the Mozilla and Netscape 6 browsers.

File Structure and Syntax

XUL files typically are denoted with a .xul file extension. The language is tag based and similar to XML; and in fact, XUL looks like a cross between XML and HTML, with some additional specialized tags. Like XML, XUL is also case sensitive.

The beginning of an XUL file could look similar to the following code:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
```

The first line denotes that this is an XML file and specifies the version, while the second line defines which style sheets will be used for this particular file. The third line designates the type of document it is, while the fourth line allocates a namespace to the XUL file listed at the end of the URL.

NOTE

Want more information on namespaces in XML? Check out <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

XUL and JavaScript

XUL by itself does not perform any specific function. For a browser to use XUL, a programming language is needed to tie the different interface components together. For the purpose of this book, JavaScript will be used as the binding language.

JavaScript can be inserted within an XUL document. To denote portions of the document that pertain to JavaScript, the `<html:script></html:script>` tag is used. Any JavaScript code is inserted between the opening and the closing tags. For more complex functionality requiring a greater deal of scripting, JavaScript code can be separated into its own file (denoted with a .js extension). This file is then included in the XUL file by specifying the `src` attribute of the `<html:script>` tag. The following shows an example of this:

```
<html:script language="JavaScript" src="myscript.js"/>
```

XUL works primarily with widgets. These are things such as menus, toolbars, buttons, scrollbars, and so on. These widgets can use JavaScript to interact with the user. For example, suppose that you create a new button widget and want to perform a specific function when the button is pressed. Using the JavaScript `onClick` event handler, you can determine when the new button has been pressed. This works for other events as well.

XUL Example

Let's look at an example of an XUL file, which creates a new menu. Listing 4.16 shows a very simple XUL program that uses the `menubar` widget.

Listing 4.16 XUL Example

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<!DOCTYPE window>
<html:script src="myScript.js"/>
<window id="new-window" title="new menus"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<menubar>
  <menu value="aNewMenu">
    <menupop>
      <menuitem value="First New Item" onclick="shoeFirst();"/>
      <menuitem value="Second New Item" onclick="showSecond();"/>
      <menuseparator/>
      <menuitem value="Last New Item" onclick="showLast();"/>
    </menupop>
  </menu>
</menubar>
</window>
```

The example creates a new menu, *aNewMenu*, and adds three options to it. A separator is used to distinguish between the first two items and the last item. Each item performs a specific JavaScript function, which is defined in a separate file named "myScript.js".

Moving On

In this chapter, we discussed a number of different items related to Client-side scripting with JavaScript. In this edition, we've expanded to new technologies including the Document Object Model and XUL. Scripting technology is becoming more advanced every day. With the advent of DOM and XML, developers are able to create a wide variety of dynamic Web content.



CHAPTER 5

JavaScript on the Server-Side

So far, you have taken an in-depth look into the JavaScript language. You have examined some of the functionality JavaScript has and some of its applications. The remaining chapters will discuss how to approach JavaScript as a programming language, and how to maximize your coding efforts by understanding the environments in which it is interpreted.

In addition to applying JavaScript on the client-side, there is also server-side usage. This feature, which is implemented in Netscape and iPlanet's Enterprise line of servers and within Microsoft's ASP environment in their Internet Information Server (IIS) servers, allows developers to build robust Web-based applications. The use of these Web-based programs can be limitless, but is most commonly used to maintain user sessions and interact with databases to build pages dynamically.



NOTE

Netscape and Sun have joined forces to jointly create various server applications under the name iPlanet (<http://www.iplanet.com>). This alliance occurred after the release of Netscape Enterprise Server 3.6, which is why the 4.0 server is referred to as iPlanet Enterprise Server. From this point on, we will simply refer to this server-side JavaScript environment as the Enterprise Server environment to avoid confusion.

In this chapter, you will take a closer look at server-side JavaScript. You'll see how it is different from client-side and learn how to deploy your applications. You will analyze the various uses of the technology and learn how to manage your

programs. Overall, this chapter will be a primer for any server-side JavaScript you might be deploying on your site.

How Does Server-Side Differ from Client-Side?

Server-side JavaScript is not only different from client-side because of the environment in which it is interpreted, but it also has divergent concepts that are applied. These distinctions can be placed into two major categories, each of which will be further broken down in this chapter. The initial categories are

- Differences in the server-side environments
- Actual programming differences from client-side

As a developer, you should be aware of these items before you begin writing server-side applications. Server-side is conceptually different from client-side, even though much of the language semantics and syntax are shared.

Server-Side Environments

One of the first things you should understand about using JavaScript on the server-side is the fact that the implementations within the Enterprise and IIS servers are different. This difference is much greater than that on the client-side, where differences appear in object models and means of implementation.

On the Enterprise front, we find that the Netscape and iPlanet alliance has created objects, properties, and methods that conform to standard server-side functionality no matter what the platform. You will find language elements that allow you to access the file system, send mail, and retrieve data from databases. None of these items are specific to any platform and are generic enough to be used across all the platforms supported by the server.

Microsoft, on the other hand, has taken a very Windows-centric approach to implementing a server-side version of its JScript language. It has many objects and related elements pertaining to the Windows environment. And lately, it has extended support for the JScript language to be included as a major language within its .NET framework. We will touch on these areas in this chapter.

Programming Differences

The next items to analyze in server-side JavaScript are the programming differences that occur. Besides the functionality added on the server-side, JavaScript also has some key programming differences. These include, but are not limited to, the following:

- Within the Enterprise Server environment, scripts to be executed on the server-side are contained between beginning `<server>` and ending `</server>` tags.
- Within the Microsoft ASP environments, the JScript language must first be specified. By default, the VBScript language is the dialect of choice.
- Additional objects, methods, properties, and events are available on the server-side.

- None of the client-side-specific language elements are available on the server-side, however the elements core to the language are.

Unlike the client-side scripts, which are contained between beginning `<script>` and ending `</script>` tags, all scripts to be executed on the server-side within the Enterprise Server are contained between beginning `<server>` and ending `</server>` tags. This tells the Enterprise server to execute all the code between these tags. As a programmer, this allows you to have client-side code on the same page without it being interpreted by the server.

On the Microsoft side of the server-side implementation, several different languages are available to the developer for writing their applications. Because of this, it is necessary to specify JScript as the language used on your page. This is accomplished in two ways.

- Including the following line as the first line on your ASP page. It sets JScript as the default language for that page.

```
<%@LANGUAGE="JScript"%>
```

- Using beginning and ending `<script>` tags with the `runat` ASP-specific attribute set to `server` and the `language` attribute set to `JScript`.

```
<script language="JScript" runat="server">
    // your code here
</script>
```

JavaScript on the server-side has objects for sending mail, opening connections to a database, managing a pool of connections, accessing the file system, and handling returned data. This syntax expansion on the server-side makes JavaScript a rich language for developing the robust applications on the server that you need.

A final difference is the fact that pure client-side language elements are not available on the server-side. This is nothing more than a reflection of the environment in which the scripts are interpreted, but it can be strange for the experienced JavaScript programmer who has not experienced the language on the server-side. For instance, there is no `navigator` or `Form` object. These are all specific to the client that is interpreting the script.

NOTE

Because there are differences in the server-side and client-side implementations, refer to Part III, "JavaScript Reference," to determine if the object, method, property, or event you want to use is supported.

Netscape's Server-Side JavaScript

Netscape was the first to implement JavaScript on the server-side, so it is no surprise that its implementation of the language builds upon the foundations developed within the browser. In this section, we will discuss what versions of their servers support JavaScript and how you run and manage applications developed in this language.

Server Support

Before you start developing any server-side JavaScript application within the Enterprise Server environment, you should first make sure of the level of support you have access to. As you are beginning to see, things have changed a lot with JavaScript over the years, and you want to make sure that you have the resources necessary to perform the tasks you need to accomplish. Table 5.1 has a list of the various versions of JavaScript on the server-side within the Enterprise Server environment.

Table 5.1 *Server-side JavaScript Support*

JavaScript Version	Enterprise Server Version
1.1	Netscape Enterprise Server 2.0
1.2	Netscape Enterprise Server 3.0
1.4	iPlanet Web Server, Enterprise Edition 4.0
1.5	iPlanet Web Server, Enterprise Edition 5.0

Compiling Your Scripts

Unlike client-side JavaScript, the server-side implementation within the Enterprise Servers require you to compile your scripts into `.web` files. These files, which are stored in byte code, contain all the files necessary for your Web-based application. The `.web` files would include any `.html` files that contain your server-side code, as well as any external source JavaScript `.js` files that you might be using.

Netscape has provided a command-line tool, the JavaScript Application Compiler (`jsac`), which is located in the `bin` directory of your installation, to compile your applications. The minimum requirement for compiling the application is the `.html` file that contains your code. The tool itself takes a variety of parameters, defined in Table 5.2, to build your `.web` file.

NOTE

In Table 5.2, all the options are listed as `-option`. The tool also accepts the syntax of `/option`. In addition, all paths should use a backslash (`\`) for directory mappings because the forward slash maps to the use of an option.

Table 5.2 *Command-Line Parameters Available to `jsac`*

Option	Description
<code>-a version</code>	This option allows you to specify the version of the interpreter against which to compile the application. At the time of this writing, this option has only one value, <code>1.2</code> , and is used to tell the compiler how to handle comparison operators, which were different in JavaScript 1.2.
<code>-c</code>	Verifies the syntax in your script, but does not create a <code>.web</code> file. Do not use this option with the <code>-o</code> option.

Option	Description
-d	Displays the JavaScript contents of your files.
-f <i>filelist</i>	This specifies a text file that contains the names of all the files you want to include in your build. The primary reason for this option is for those operating systems that have a limit to the number of characters you can specify on a single command line. Each filename in the file should be separated by a space. If your filename has a space in it, include the name within quotes.
-h	Displays help for the tool. Do not use with any other options.
-i <i>inputfile</i>	Specifies the name of a single input <code>.html</code> file. See <code>-f</code> when including multiple files.
-l <i>characterset</i>	Allows you to specify the character set used when compiling your application. This would be something similar to <code>iso-8859-1</code> .
-o <i>outputfile</i>	Specifies the name of the output <code>.web</code> file you are creating. Do not use this with the <code>-c</code> option.
-p <i>path</i>	Designates the root for any relative path filenames you specify.
-r <i>errorfile</i>	This option allows you to specify a file to which to have all errors written.
-v	Displays verbose information about the execution of the compile.

Using these options, a typical build of an application might look something similar to the following.

```
jsac -o MyApp.web -v index.html results.html jsbeans.js
```

It specifies to display the build process verbose, defines the output file as `MyApp.web`, and gives the filenames of the files to include.

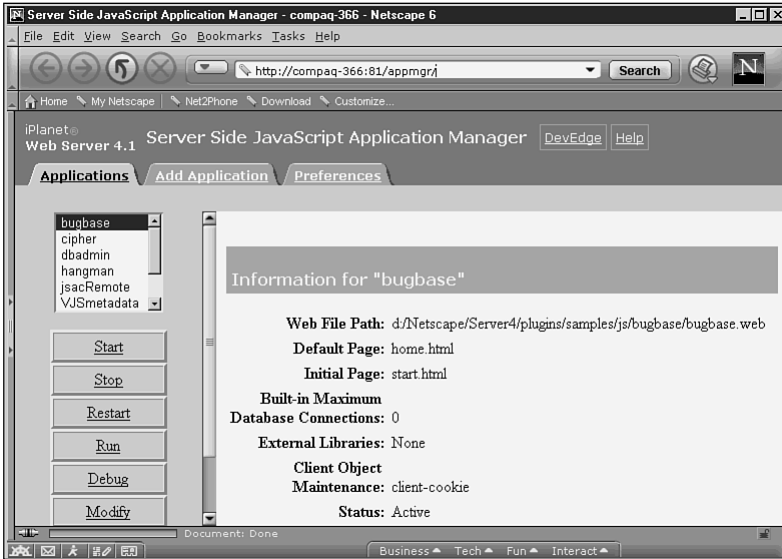
Here is another example that specifies a full path to the input file as well as sets an error file to log any errors during the compilation process.

```
jsac -o MyApp.web -v -i /js/myapps/index.html -r /js/logs/myapperror.log
```

As many programmers know, it is very important to know your command-line options. Knowing how to troubleshoot by compiling a project can often be just as challenging as creating one.

The Server Side JavaScript Application Manager

The Server Side JavaScript Application Manager, shown in Figure 5.1, is the program used to manage server-side applications within the Enterprise Server environment. From within this common interface, you can start, stop, restart, run, debug, modify, and remove applications created for a Web site. It is a dashboard to all the server-side JavaScript applications installed.

**Figure 5.1**

Enterprise Server's Server Side JavaScript Application Manager.

This manager is much more than a start and stop location for applications. In fact, it is most useful for verifying current status and information, debugging, and setting configuration options. After an application has been compiled and added to the Server Side JavaScript Application Manager, this interface is used to maintain all aspects of the program—not including the actual coding, of course. However, before you can use the JavaScript Application Manager, you must do a few things.

Configuration

Before the Server Side JavaScript Application Manager is available for use, it must be enabled within an instance of Enterprise Server. This can be done easily from the Server Manager.

Once logged in to the Administration Server, simply select the option that represents the instance of the Web server on which you want to enable server-side JavaScript. When you have accessed the Server Manager for this instance, click the Programs button in the menu bar. Next, click the link to Server Side JavaScript in the navigation bar to the left of the page. This takes you to the main screen, shown in Figure 5.2.

When you have access to this page, there are three options:

- Activate the server-side JavaScript environment
- Require a password for access to the JavaScript Application Manager
- Select a JavaScript Thread Pool option



Figure 5.2

Enabling server-side JavaScript through the Server Manager.

The first thing you need to select in this screen is to activate the server-side JavaScript environment. Next, consider using the Administration password to access the JavaScript Application Manager. This is highly recommended because it is a major security risk to unprotect applications. Without the password, a person could stop, or even worse, modify the settings for applications. And finally, if you plan on using server-side JavaScript to connect to databases, you can specify a thread pool. For more information on thread pooling, click the Help button on this screen.

When server-side JavaScript is enabled and you have made a decision on whether to password protect the application manager, click the OK button. This will take you to the Save And Apply Changes page, where you should click Save And Apply to activate these options. An alert box will be displayed, confirming a successful or unsuccessful change.

PROBLEMS?

If problems are encountered here, check the following:

- Are any NSAPI (Netscape Server Application Programming Interface) plug-ins running that might conflict with server-side JavaScript?
- Is any kind of third-party software being used to dynamically build the Web pages for this site that might have conflicting or overlapping settings?
- Has someone hand modified the obj.conf file recently?

For the first two, think specifically about those applications or NSAPI plug-ins that parse pages. Consider turning off Parsed HTML if it is enabled. This can be done within the Web server instance's administration by clicking the Content Management menu item and then clicking the Parse HTML link in the navigation.

Finally, if there are still troubles or if the third bullet point came into play, consider loading a previous version of the `obj.conf` file. This can be done by clicking the Server Preferences menu item, and then selecting the Restore Configuration link in the navigation. Check under the `obj.conf` column and roll it back one version at a time until it works. Note that you are undoing any prior changes of the configuration file. Doing so might break another process while trying to fix the JavaScript ApplicationManager.

When a successful saving of the configuration file (`obj.conf`) and restarting of the Web server has been accomplished, you will be returned to the Active Server Side JavaScript page. At the top of the page, there will now be a link to the Server Side JavaScript Application Manager. This link will be in the following form, where `machinename` is replaced with the name of your machine and domain if applicable.

`http://machinename/appmgr`

Now that everything is configured to run the application manager, click the link. This will launch another page and load the Server Side JavaScript Application Manager. When on this page, click the Preferences tab. This will display a page to set up some defaults when adding new applications and set a couple of preferences.

Table 5.3 defines the options available when setting the default values for new applications.

Table 5.3 *Default Application Settings That Can Be Set for the JavaScript Application Manager*

Setting	Description
Web File Path	This is the default path to your <code>.web</code> file. If you store all your applications in a subdirectory of <code>/ssjsapps/myprojects</code> , you should choose this option.
Default Page	This option specifies the default page of your application. For practical purposes, you might want to set this to <code>index.html</code> .
Initial Page	This specifies the initial page of your application. For example, if your application has global settings that are only loaded when the application is first started and you have stored this in a file called <code>global.html</code> , you should set this option to this file.
Built-In Maximum Database Connections	Those of you who are using databases that charge on a per connection basis will like this feature. It is the default that restricts your applications to a set number of maximum connections.

Setting	Description
External Libraries	This contains the absolute path to any external libraries that your application might be using.
Client Object Maintenance	This default option is used to determine if you are going to maintain user sessions in your application. The possible choices are Client Cookie, Client-URL, Server-IP, Server-Cookie, and Server-URL.

After specifying the options for the Default Values When Adding Applications section, specify your preferences. Within the Preferences section, there are two items to set:

- *Confirm On*—This option determines if a confirmation dialog box is to pop up before you perform a task. The list of tasks this can be enabled for are Remove, Start, Stop, and/or Restart an application.
- *Debug Output*—This option allows you to choose if you want to do your debugging in the Same Window or Other Window of the application.

After these settings are completed, click the OK button. This will finish the configuration of the Server Side JavaScript Application Manager. Additional help or documentation can be accessed by clicking the links within the user interface.

Script Management

Managing scripts might be one of the most overlooked aspects of deploying a Web-based application. The mentality seems to be, “It was written to never break.” This, as we all know, simply never holds up in the real world. Even applications that worked perfectly for years will eventually hit a bump. It might be after an upgrade to a new server, or maybe a new browser has come out that implements some client-side JavaScript differently than expected. Either way, it will happen, so expect it.

The Server Side JavaScript Application Manager provides a common location to manage applications as well as perform maintenance on them. You can access the online documentation at DevEdge and launch the help window with documentation before clicking any of the available tabs.

When the Applications tab is selected, you will see the list of applications currently installed. These are located in a scrolling text area at the top of the section. After an application is selected, any of the options defined in Table 5.4 can be used.

Table 5.4 Application Options in the Controls Section

Control Option	Description
Start	Starts the application and makes it available for your users to use.
Stop	Stops the selected application.
Restart	Stops and restarts the selected application.
Run	Launches a separate window and loads the URL to the selected application. This allows you to quickly test your application to ensure that it is working properly.

Table 5.4 Continued

Control Option	Description
Debug	Launches either one or two windows, depending on how you set your preferences, to debug your application.
Modify	Allows you to modify any of the settings you have for that application.
Remove	Removes the application from your list of managed applications.

This section discusses the ability to manage applications. When an application is selected, its information is displayed as well as its current status. Now take a look at using the Debug feature, which is a valuable tool for any server-side JavaScript developer.

To launch the Debugging window, select the application to debug, and then click the Debug button. Depending on how the preferences are set, you will see one or two main windows. If there is only one window, it will be split into two frames.

One of these elements is the Debugging window, seen as a single window in Figure 5.3. As a program runs, the contents of this window will change as JavaScript code is processed within the application. This tool provides a look at the information being passed back and forth. It is possible to see variables being set, as well as objects created and their various states.

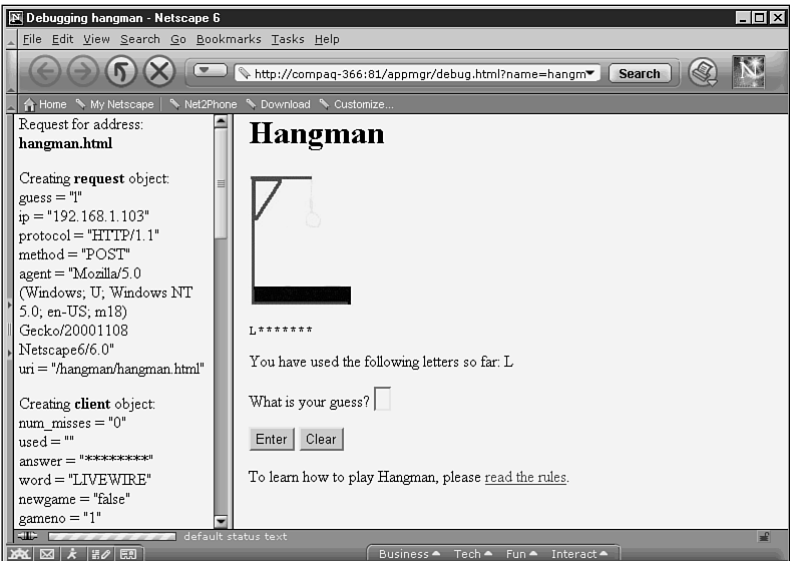


Figure 5.3

The Debugging window is used to display various bits of information about the application being debugged.

As a programmer, you often will want to know the value of variables you create and return values of methods. Server-side JavaScript makes this possible by using the `debug()` method in scripts to write information to the Trace Information window during runtime. This method is similar to the `write()` server-side method, where a string or variable is passed, and the interpreter writes the data passed to the window. A typical use of this method might look similar to the following:

```
debug("myFunction entered and was passed: " + myVar);
```

NOTE

For more information on using the `debug()` method, see its entry in Chapter 9, "Server-Side."

In addition to knowing how to debug applications, it is important to understand the error messages you might run across. There are two main types of errors. One type of error, which is common for syntax mistakes, is a standard server-side JavaScript error. This error will return the object in which the error occurred, the error message, the filename of the application, and the line number.

The second types of errors are those generated by databases. These errors can have several different types of information returned, but usually contain the following:

- *Database/Function*—This error alerts the programmer that a database error occurred, and it lists the function in which it occurred.
- *LiveWire Services Error*—This error, generated by the LiveWire database connection service, states that an error occurred and lists a message as to why.
- *Vendor Error*—This error is also returned by the LiveWire service, but it reports an error specific to the database to which you are connecting. You will need to know your database error codes to decipher this error.
- *Standard Server-Side JavaScript Error*—This error is the same as the first type, except that it is specific to your use of a database connection.

In addition to the Debugging window, there is a main window, or frame. When debugging an application, this window contains the application itself. The combination of this window and the Debugging window allows a developer to step through, as a user would, and see what is going on in the background. With the Debug feature of the Server-Side JavaScript Application Manager, a developer can correct and work through almost any problem he might encounter.

Microsoft's ASP Environment

Like Netscape, Microsoft soon saw the value in implementing JScript on the server-side and within its .NET framework. The ability to give developers a single language for both client-side and server-side was too compelling. So, with the release of IIS 3.0, ASP 1.0 was made available, and within that release, JScript was at the disposal of developers. Since that time we have seen the release of IIS 4.0 and 5.0, along with ASP 2.0, 3.0, and ASP+ (now referred to as ASP.NET).

Server Support

Before you start developing any server-side JScript application within the ASP environment, you should first make sure of the level of support you have access to. Table 5.5 has a list of the various versions of JScript on the server-side within IIS.

Table 5.5 *Server-side JScript Support*

JScript Version	IIS Support	ASP Version
2.0	3.0	1.0
3.0	4.0	2.0
5.0	5.0	3.0
6.0	6.0	ASP.NET (previously known as ASP+)

Using the JScript Language

Before we go any further, there is one interesting thing to note about Microsoft's implementation of JScript. Unlike, at least currently, other browsers and environments that support an ECMAScript-based scripting language, the JScript engine within Microsoft's applications can be upgraded. This means that you might only be currently running IIS 4.0, but it is possible to download and install the JScript 5.6 engine.

Actually using the JScript language is fairly easy. As we touched on in Chapter 1, "What Is JavaScript to a Programmer?" there are a few extra objects, listed in Table 5.6, that you need to be aware of. Other than these, you can use the normal objects, properties, and methods you are accustomed to using.

Table 5.6 *Server-side JScript Objects*

Object	Description
Application	Contains elements that allow you to share information across all users of a given application.
ASPErrors	Contains elements that allow you to obtain information on errors that might have occurred within your ASP application.
ObjectContext	Contains elements that allow you to either commit or abort a transaction that has been initiated by your script.
Request	Contains elements specific to the browser making the request to IIS.
Response	Contains elements specific to the server returning a request to the user's browser.
Server	Contains elements specific to the environment in which the script is running. This includes access to the various environment variables.
Session	Contains elements specific to a user's session.

 NOTE

See Chapter 9 for more information on these objects and their properties, methods, events, and collections.

Checking Your Environment

Before you start trying to build ASP applications using JScript, there are a few items that you should be aware of. These items might differ slightly across the various versions of IIS, but conceptually they are consistent.

- Make sure that your installation of IIS is configured to load and execute .asp files with the asp.dll.
- Make sure that your Home Directory permissions allow for Script or Execute permissions.
- Make sure that you have either explicitly set JScript as the language on your pages, or that you have set it as the default language within the properties of your instance of IIS.

Although these items seem minor, failure to have these set can prevent your scripts from running. By default, everything, with the exception of the default language being JScript, should be set up correctly. To test, you can use the code in Listing 5.1. If you are able to serve the page through IIS to a browser and see "It Worked!", you will be able to proceed with the ASP examples later in this chapter.

Listing 5.1 Sample Code to Test if ASP Is Properly Functioning Within Your Instance of IIS

```
<html>
<head>
  <title>JScript within ASP Test</title>
</head>
<body>
  <script language="JScript" runat="server">

    // Write "It Worked!" to the users page.
    Response.Write("It Worked!");

  </script>
</body>
</html>
```

Uses of Server-Side JavaScript

Now that you have seen some of the differences of server-side and client-side JavaScript and how to create your applications within the appropriate environments, take a look at some of Server-Side JavaScript's uses. This section does not try to

outline all the uses you might encounter, but rather discusses some of the more common aspects of using server-side JavaScript. We have included examples for both environments specific to the server-side implementation, so this should give you a good taste of what you can do with this powerful language.

Within Enterprise Server

Because both the Enterprise and IIS environments differ as do their implementation of the JavaScript language, we will first take a look at examples specific to Enterprise Server. In this section, we will explore the following:

- Database connectivity
- E-mail
- Working with Files

Database Connectivity

One of the major features of server-side JavaScript is its capability to connect to databases. As you know, Netscape has paved the way for the use of standardized technology, and you can expect nothing less when connecting to databases. It provides native-support, industry-leading databases such as DB2, Informix, Oracle, and Sybase databases, as well as access through ODBC for other databases, such as Microsoft's SQL Server.

The functionality server-side JavaScript provides for connecting to databases is done through its LiveWire service, which no longer has to be installed as a separate component. This service provides JavaScript objects that can be used to connect to various databases. Once connected, you are able to run the SQL statements necessary to perform the operations you want.

NOTE

When Netscape released version 2 of Enterprise Server, LiveWire was a blanket term for "server-side JavaScript." By the time version 3 of Enterprise Server was released, the acceptance of JavaScript as an industry standard had increased. To maintain some consistency across terminology, the term LiveWire has now, starting with version 3 of Enterprise Server, been associated with the service that allows developers to write code to interact with databases.

Connections to a database are maintained in a *pool*. Before you open a connection to a database and have the ability to run queries against it, you must create an instance of the DbPool object. When the instance is created, you can obtain connections from the pool as needed. The pool object itself takes all the parameters necessary to make the connection. It is possible to create a pool without specifying any parameters. However, you must pass the parameters when the first connection is attempted.

The following code is the syntax for creating a DbPool object. Each parameter is defined in the bulleted list following the example.


```
var myPool = new DbPool (DBType, DBInstance, UID, PWD, DBName, MaxConn,
    ➤CommitFlag);
```

- **DBType**—The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
- **DBInstance**—This is the instance name of the database. For ODBC it is the DSN entry name.
- **UID**—The username or ID you want the connections to connect as.
- **PWD**—The password for the user you are connecting as.
- **DBName**—The name of the database into which you want to log. For Oracle, DB2, and ODBC connections, this should be a blank, "", string. In Oracle, the name of the database for these connections is set up in the `tnsnames.ora` file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the **DBInstance**.
- **MaxConn**—The maximum number of connections to the pool. This is effectively the number of connections the pool will open to the database.
- **CommitFlag**—This flag determines if a pending transaction is committed when the connection is released. If it is set to `false`, the transaction is rolled back. If it is set to `true`, it is committed.

Because it is possible to create an instance of this object by passing a limited set of these parameters, as well as passing none, you should reference the `DbPool` entry in Chapter 9 before using this object.

After you have created a pool, you can use the connections within that pool as needed. To pull a connection, use the `connection()` method on your pool. This will return an available connection to use for processing. The syntax and a description of the parameters are as follows:

```
var myConn = myPool.connection(name, timeout);
```

- **name**—This is a name you can give your connection. Because you actually store the connection in a variable, this name's primary function becomes one for debugging purposes.
- **timeout**—A numeric value for the number of seconds you give the instance to connect.

After the connection has been made, you are able to perform the necessary processing you require for your application. For more information on the methods available, check entries for the `Connection`, `Cursor`, and database objects in Chapter 9.

USER SESSIONS AND DATABASE ACCESS

An extensive discussion of how to manage user sessions and database connections as various users access your application is beyond the scope of this book. However, there are some items you should be aware of when performing these tasks.

You should know how database queries are handled. You should understand how the information is returned from your specific database and how to process the information.

You should also know how to appropriately manage user connections to your application. It is possible your application will have to manage many connections from many users at the same time. If your sessions and connections get crossed, you might send the wrong information to the wrong user.

Maintaining and managing your user sessions and database connections can be a very important factor and you should take great care when working with them.

E-Mail

Another feature that can be exploited and implemented within server-side JavaScript applications is the ability to send mail. The properties and methods needed to perform these tasks are contained in the `SendMail` object.

NOTE

Refer to the Chapter 9, "Server-Side," for more information on the `SendMail` object.

Using the `SendMail` object is very straightforward. Simply set the same properties contained in the everyday e-mail you send and invoke the `send()` method. If an error is encountered, it can be analyzed by using the error methods supplied. Listing 5.2 shows an example use of this object to create a page for users to send e-mail. Figure 5.4 shows what is displayed to the user when she encounters this page, and Figure 5.5 shows the results of submitting a successful e-mail.

Listing 5.2 *Example of Using the `SendMail` Object*

```
<html>
<head>
  <title> Using the SendMail object</title>
</head>
<body>
<server>

  // See if they have submitted or just need the form.
  if(request.method == "POST"){

    // Create an instance of the SendMail object.
    var myMail = new SendMail();

    // Assign the properties their values.
    myMail.To = request.toAddress;
    myMail.From = request.fromAddress;
    myMail.Subject = request.subject;
    myMail.Body = request.body;
    myMail.Smtpserver = "mail.purejavascript.com";
    myMail.Errorsto = "errors@purejavascript.com"

    // Try to send the mail.
    if(!myMail.send()){
```

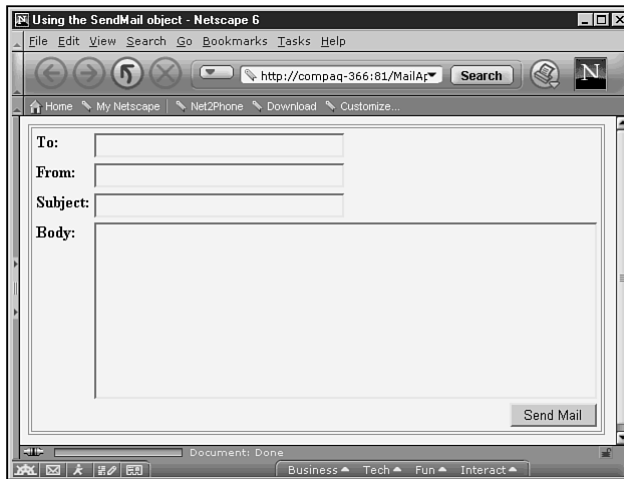
```

// If there was an error, give the user the e-mail address of who they
// should contact about the error, as well as the error code and
// message.
write("Error sending your message. Please send e-mail to ");
write(myMail.Errorsto + " with the following error message");
write("Error " + myMail.errorCode() + ": " + myMail.errorMessage());
}else{

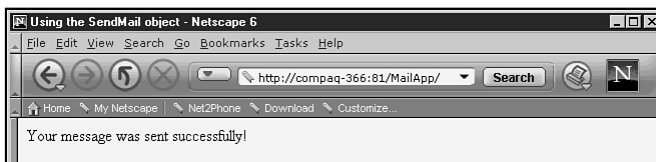
    // If there was not an error, tell the user they were successful.
    write("Your message was sent successfully!");
}
}else{

    // If this page was called and a form was not submitted, write the
    // email form to the page for the user to use.
    write('<form name="myForm" method="post">');
    write('<table border="1"><tr><td>');
    write('<table border="0">');
    write('<tr align="left" valign="top">');
    write('<td><b>To:</b></td>');
    write('<td><input type="text" name="toAddress" size="30"></td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td><b>From:</b></td>');
    write('<td><input type="text" name="fromAddress" size="30"></td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td><b>Subject:</b></td>');
    write('<td><input type="text" name="subject" size="30"></td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td><b>Body:</b></td>');
    write('<td><textarea name="body">');
    write(' cols="60" rows="10" wrap="soft"></textarea>');
    write('</td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td colspan=2 align="right"><input type="submit">');
    write(' value="Send Mail">');
    write('</td>');
    write('</tr>');
    write('</table>');
    write('</td></tr></table>');
    write('</form>');
}
</server>
</body>
</html>

```

**Figure 5.4**

Building an e-mail page for your applications.

**Figure 5.5**

The results of submitting an e-mail successfully.

As the example demonstrates, the `SendMail` object makes it easy to create a page through which users can send mail. In a true, fully developed, Web-based application, a programmer should add code to check for errors in the submission. This would be an appropriate time to use client-side JavaScript to ensure that basic requirements, such as syntactically correct e-mail addresses, are entered.

Working with Files

The `File` object allows you to perform various system tasks, such as reading and writing to a file on your disk. The `File` object itself has many methods and a prototype property that allows a programmer to create new properties and methods of the object.

NOTE

Refer to Chapter 9 for more information on the `File` object.

Similar to the `SendMail` object, the use of the `File` object is straightforward. The methods provided allow you to perform the various tasks needed on the files in your file system.

Part of the functionality of working with these files allows a programmer to specify how he wants to open the file. A file can be opened to read, write, append, or open in binary mode. These options are specified in the `open()` method in the following form.

```
myFile.open( "option" );
```

NOTE

Refer to Chapter 9 for more information on the options that can be passed to this method.

In your applications, you might want to display the contents of a file. This program could be an administration application that reads a file and displays its contents on a page. Listing 5.3 contains an application that displays the contents of a selected log file on the file system.

Listing 5.3 Using the `File` Object to Display the Contents of a File

```
<html>
<head>
  <title> Using the File object</title>
</head>
<body>
<server>

// See if they have submitted or just need the form.
if(request.method == "POST"){

  // Create an instance of the File object and pass it the file
  // the user specified they wanted to view.
  var myLog = new File(request.file);

  // Try to open the file.
  if(!myLog.open("r")){

    // If there was an error, tell the user.
    write("There was an error opening the file: " + request.file);
  }else{

    // If there was not an error, then open the file and display it.
    write('<h3>The contents of ' + request.file + ' are as follows:</h3>');
    while(!myLog.eof()){
      write(myLog.readLine());
    }
  }
}else{

  // If this page was called then write the select box to the page for
  // the user to use select which log they want to see.
```

```
    write('<form name="myForm" method="post">');
    write('<select name="file">');
    write('<option value="/logs/admin.log">Admin Log</option>');
    write('<option value="/logs/user.log">User Log</option>');
    write('<option value="/logs/error.log">Error Log</option>');
    write('</select>');
    write('<input type="submit" value="View Log">');
    write('</form>');
}
</server>
</body>
</html>
```

In this example, a user has the ability to select a file for viewing. After submitting the request, the example tries to open the file for reading. If it is unsuccessful, an error with the filename is returned to the user. If the file opens, the contents are displayed in the document.

Within IIS

On the IIS side of the fence, it is important to show how the language syntax differs from Enterprise Server's implementation, but remains true to the JavaScript language. Within this section we will explore examples in the follow areas:

- Form submissions
- Session handling

Form Submissions

One of the nice things about JavaScript on the server-side within both the Enterprise and IIS environments is the ease with which you can process form data. Within IIS this is accomplished through the use of the Request object and its associated properties, methods, and collections, which are detailed in Chapter 9.

To use this object, we must create a form that will pass some information. For simplicity's sake, we will prompt the user for his first and last name. Listing 5.4 contains the HTML for this page.

Listing 5.4 HTML for Our Form

```
<html>
<head>
  <title>ASP Form Example</title>
</head>
<body>
  <form action="process.asp" method="post">
    First Name:<input name="first" width="10"><br>
    Last Name:<input name="last" width="10"><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Now that the form has been created, we must create the `process.asp` file to handle its submission. Within this file, we will use the `Write()` method of the `Response` object to output the values of the passed information. As we mentioned before, JScript has several objects specific to their IIS implementation for processing data. The `Response` object, which is detailed in Chapter 9.

We will use the `Request.Form` collection, which takes the name of the form element you are trying to access. In our example, this will be for `first` and `last`. Listing 5.5 shows the code we can use to complete this task.

Listing 5.5 ASP Code Using JScript to Process Our Form

```
<html>
<head>
  <title>ASP Form Example: Results</title>
</head>
<body>
<script language="JScript" runat="server">

  // write the first and last name entered in the form
  Response.Write("You entered (last, first format): ");
  Response.Write(Request.Form("last") + ", " + Request.Form("first"));

</script>
</body>
</html>
```

Server Variables

Another use of ASP is to access and use various server variables. For instance, on many occasions a Web developer might need to determine the browser making the request. He might want to direct the user agent to a specific site, or possibly post a warning message.

In Listing 5.6 we use the `Request.ServerVariables` collection to access the `HTTP_USER_AGENT` variable. After loading into a variable, we use the `indexOf()` method to see if it is Internet Explorer. We then print out a message stating if it is Internet Explorer or not followed by the entire user-agent string.

Listing 5.6 Using the Request.ServerVariables Collection

```
<html>
<head>
  <title>ASP Form Example: Results</title>
</head>
<body>
<script language="JScript" runat="server">

  // store user agent string in variable
  var ua = new String(Request.ServerVariables("HTTP_USER_AGENT"));
```

```
// Check to see if its Internet Explorer.
if(ua.indexOf("MSIE") != -1){
    Response.Write("Welcome!<br />");
    Response.Write("User-agent: ");
    Response.Write(Request.ServerVariables("HTTP_USER_AGENT"));

    // Browser is not IE, so do something different.
}else{
    Response.Write("You are not using an Internet Explorer browser<br />");
    Response.Write("User-agent: " +
Request.ServerVariables("HTTP_USER_AGENT"));
}

</script>
</body>
</html>
```

Moving On

Up until this point, we have covered the use of JavaScript on both the client and server-side. For many of you, this represents the borders of what can be accomplished with JavaScript. However, Chapter 6, “Windows Scripting,” will map out yet another environment where JavaScript (or JScript rather) has taken hold—the Windows Script Host environment.



CHAPTER 6

Windows Scripting

As many developers know, a big deficiency with Windows was its lack of scripting support for automating tasks. Most other operating systems had some type of built-in scripting engine. Of course there was the batch file support in MS-DOS, but it was very limited in its capability and not very useful in the Windows environment.

Microsoft realized this limitation and introduced the Windows Script specification within Internet Explorer 3.0. This provided an interface for developers to build scripting engines for different browsers. From the Windows Script Interfaces grew what is known today as the Windows Script Host.

Windows Script Host

Windows Script Host (WSH) provides the ability to automate tasks for the Windows environment. Before WSH, batch files were available in MS-DOS. However, they weren't very powerful and provided little use in the Windows environment. A scripting language was needed for Windows, and WSH was the answer. Suppose that you wanted to get the computer name, add a desktop shortcut, and map a network drive without doing each task individually. With Windows Script Host, all these tasks can be performed through a single script.

WSH can be used with a few different programming languages. VBScript and JScript are supported in the default engine on Windows 98 and NT/2000. This book focuses on the JScript language and will use it in the examples.

The latest version of WSH is 5.6. Although it is still a beta version, the scripting engine can be downloaded from Microsoft's Web site. The added benefits of this version are

improved argument handling, remote script capability, treating processes as objects, access to the current working directory, and an improved security model.

The way Windows Script Host works is you create a script using various methods and properties from the WSH objects and save it to a file with the .wsf extension. This specifies that the file is a Windows Script Host file. Let's take a look at an example of a simple WSH file. Listing 6.1 shows a WSH file that will display the username of the local machine in which the script is run.

Listing 6.1 Example of a WSH File

```
<job>
<script language="JScript">

var WshNetwork = WScript.CreateObject("Wscript.Network");
WScript.Echo("The User name is: " + WshNetwork.UserName);

</script>
</job>
```

The syntax of a WSH script is fairly simple. You enclose your functionality between the opening and closing <job><script> tags. This is similar to a normal JScript or JavaScript program. This script can be run a few different ways. These ways are discussed later in the chapter.

Architecture

The architecture of the Windows Script Host is fairly basic. There are only a few components, which operate together, to make WSH scripts come alive. They are the script itself, the Windows Shell, the Windows Registry, and the WSH engine. Figure 6.1 shows a diagram indicating how the components interact with each other to process the script.

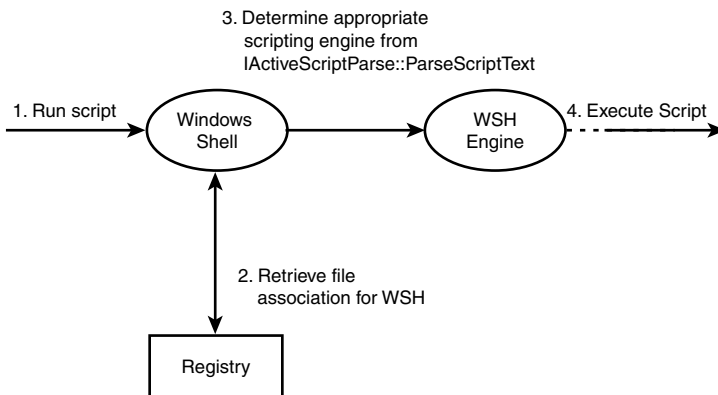


Figure 6.1

WSH architecture.

As you can see, the Windows Shell is the first component to encounter the script file. The shell must then determine what this file is and does so by looking up the file association in the Registry. When it has this information, it can then pass the file to the Windows Script Host Engine, which will use the `IActiveScriptParse::ParseScriptText` method to determine whether this particular script will go to a JavaScript engine, Visual Basic Script engine, Perl engine, and so on.

Object Model

The WSH is based on an object model hierarchy, which consists of 14 objects. Every object stems from the `WScript` object (root object). The WSH Object model is shown in Figure 6.2.

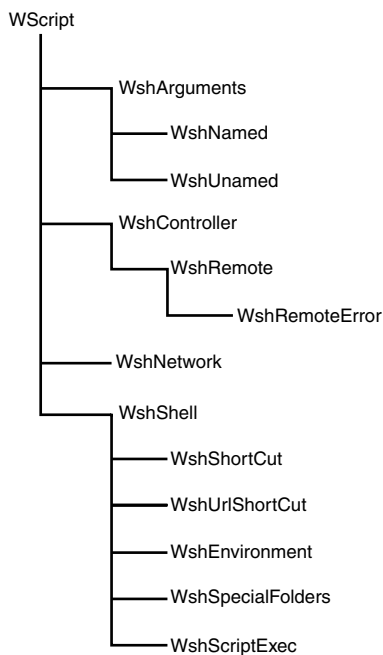


Figure 6.2

WSH object model hierarchy.

Not every object can be instantiated directly. The `WScript` object is available in every WSH script and does not need to be instantiated. Other objects that can be instantiated are the `WshController`, `WshNetwork`, and `WshShell`. To create these objects, use the `CreateObject` method. All other child objects are indirectly created through the top-level objects. The WSH object model exposes various COM objects, which can be manipulated.

Benefits and Limitations

WSH has both its benefits and limitations. Using JScript or VBScript, WSH provides you direct access to a computer and its Windows resources. You can directly manipulate the Windows Registry, set default printers, run remote scripts and more. The WSH is a low memory scripting host that works well with scripts that require little user interaction. So for things such as entering logon scripts and batch processes, and performing administrative tasks, WSH is a great tool to use.

However, with its power comes disadvantages. Providing a script direct access to your local computers' resources can be very dangerous. Should a developer decide to write a malicious script, a lot of damage can be caused. For this reason, some choose for their Windows environment to disable WSH.

WSH is limited in some aspects. The graphical components are limited. There are no custom dialogs or anything of that sort. For better UI's, your program will need to rely on the tools available with whichever language you choose to write your script in.

Windows Script Components

Many different components are tied together to create the whole Windows Script Host object model. In this section we will discuss some of these objects and their capabilities in more detail.

Objects

In the Object Model, you saw that many different objects comprise the Windows Script Host. Each object has its own methods and properties associated with it. The main objects from which all other minor objects stem are WScript, WshController, WshNetwork, and WshShell.

The three most commonly used ones that we'll discuss are WScript, WshShell, and WshNetwork. They provide the majority of the top-level functionality for Windows Script Host. Let us take a look at what each of them can do.

WScript

The WScript object is the root-level object from which all other objects are created. It does not need to be instantiated and is available from any script file. It can be used for a number of different tasks. You can use it as an informational object to get script file-names, command-line arguments, and host information. It can also be used to create objects, connect to and disconnect from objects, sink events, stop script execution, and output information. Listing 6.1 displays the simplest form of using the WScript object by using its Echo method to output the username.

As mentioned earlier, WScript can create any type of child object by using the CreateObject method. Suppose that you wanted to create a new WshShell object. You could do so by using the following line.

```
WshShellObj = WScript.CreateObject("WScript.Shell");
```

By now you are probably saying to yourself, “This is great, but what can I do with a new WshShell object?” Keep reading and you will find out. (See Chapter 15, “Windows Script Host,” for details on all the methods and properties associated with the WScript object.)

WshShell

WshShell is a very useful object. It is used for manipulating the Windows Registry, creating shortcuts, starting new shell processes, and reading system information. It provides the ENVIRONMENT collection for your program, which allows you to access or manipulate environment variables. (See Chapter 15 for details on all the methods and properties associated with the WshShell object.)

Let’s look at a simple example in Listing 6.2, which creates a shortcut on the desktop to the Amazon.com Web site. Using the CreateObject method of the root WScript object, we can create a WshShell object named WshShellObj. Next we need to specify that the shortcut is to be created on the desktop. To accomplish this, the SpecialFolders method is used with “Desktop” as the parameter. Then we call the CreateShortcut method and pass it the path for our shortcut. Finally the target path for the shortcut is specified and the shortcut is saved.

Listing 6.2 *Example of Creating a Shortcut with the WshShell Object*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Specify the Desktop as a folder.
myDesktop = WshShellObj.SpecialFolders("Desktop");

// Create a shortcut to the Amazon.com website.
var urlShortcut = WshShellObj.CreateShortcut(myDesktop + "\\Amazon.url");
urlShortcut.TargetPath = "http://www.amazon.com";
urlShortcut.Save();

</script>
</job>
```

WshNetwork

The WshNetwork object is used to perform many different network-related tasks. With this object, you can add printer connections, obtain the computer name, map network drives, set default printers, get user domains, and more. These types of tasks can be very useful to a Windows network administrator. (See Chapter 15 for details on all the methods and properties associated with the WshNetwork object.)

Let’s look at an example. Suppose that a new network printer has been added and the network administrator wants to make it as easy as possible for users to add this printer

to their machine. The administrator could create a WSH script that all users can run to add the printer to their machines. In Listing 6.3, we see the code to add a new printer.

Listing 6.3 *Adding Printer with WshNetwork Object*

```
<job>
<script language="JScript">

// Create Arguments object.
WshArgObj = WScript.Arguments;

// Read in the username and password.
var userName = WshArgObj.Item(0);
var passwd = WshArgObj.Item(1);

// Create a new Network object and add printer.
var WshNetwork = new CreateObject("Wscript.Network");
var printer = "\\printers\NewPrinter";
var localName = "myNewPrinter";
WshNetwork.AddPrinterConnection(localName, printer, userName, passwd);

</script>
</job>
```

Because user input is required for this particular script, it is recommended that the command line method, `cscript.exe`, is used to execute the script. (For more information on differences between `cscript.exe` and `wscript.exe`, refer to the “Methods of Executing Script” section). The user would specify his username and password as the arguments. So assuming that the file is named “`addPrinter.wsf`”, it would be executed in an MS-DOS Windows as follows:

```
cscript addPrinter.wsf Betty pass123
```

Methods of Executing Scripts

To run a WSH script, you need to have the Windows Scripting Engine installed on your operating system. This is included in Windows 98 and NT/2000. For Windows 95 users, the WSH engine can be added by downloading the engine from the Microsoft site and installing it. See <http://msdn.microsoft.com/scripting> for more information.

Once installed, WSH scripts can be run in two different modes. They are Windows-based and DOS-(command) based. The mode in which you run your scripts depends on what tasks you intend to accomplish. If you intend to have some user interaction and want to use graphical components, the Windows-based execution is preferred (`wscript.exe`). If your script needs to perform tasks, which can only be done in MS-DOS or require little user interaction, the command-based execution should be used (`cscript.exe`). Both methods of execution can be run from a command line and have certain options that can be specified. They are listed in Table 6.1.

Table 6.1 *wscript.exe and cscript.exe Options*

Type	Description
//B	Specifies the batch mode. Suppresses user prompts and script error messages.
//D	Enables the debugger.
//E:engine	Runs the script with the specific script engine.
//H:cscript	Uses cscript.exe as the default execution method.
//H:wscript	Uses wscript.exe as the default execution method.
//I	Specifies the interactive mode. Displays user prompts and script errors.
//Job:<jobID>	Executes a specified jobID from the .wsf file.
//logo	Displays an execution banner.
//nologo	Prevents display of the execution banner.
//S	Saves current command-line options.
//T:nn	Sets the maximum number of seconds that a script can run.
//U	Not available with wscript.exe.
//X	Launches a program in the debugger.
//?	Help. Displays information for command parameters.

To execute a script generically from the command line, adhere to the following syntax:

```
wscript <filename> [//options] [arguments]
cscript <filename> [//options] [arguments]
```

Arguments should be separated with a space. WSH also supports drag-and-drop functionality for arguments. You can drag and drop a file onto a WSH file, and—provided that your script is written to parse the argument—it will treat your dragged filename as an argument.

Windows-Based Scripts

Windows-based WSH scripts allow you to take advantage of the limited graphical capabilities in WSH. This is particularly useful if your script requires visual user interaction. Keep in mind though that WSH doesn't have very complex dialog boxes. The graphical components are mostly limited to the tools available in VBScript or JScript and pop-up dialogs.

There are three methods to execute a Windows-based script. The easiest is to locate the script file in Windows Explorer and then double-click it to start. The second method is to use the "Run" option from the Windows Start Menu. Choose the WSH file in the "Run" option and then click the OK button to run. This is shown in Figure 6.3. The third method is to run your script from the command line using wscript.exe.

Any of these methods will work, and it is just a matter of personal preference. Let's take a look at the example in Listing 6.4.

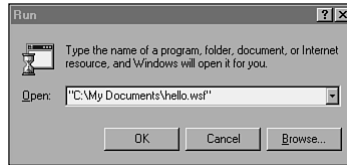


Figure 6.3

Running script from the Run option.

Listing 6.4 *Simple Hello World Script*

```
<job>
<script language="JScript">

// Output the Hello World text.
WScript.Echo("Hello World!");

</script>
</job>
```

If we were to run the script as a Windows-based script, the output would display a pop-up window as shown in Figure 6.4.



Figure 6.4

The WSH pop-up display box.

Command-Based Scripts

WSH scripts can also be run from the command prompt. This is performed in the DOS environment using the `cscript` executable. For example, if you wanted to run a script called `hello.wsf`, you would type the following at a command prompt:

```
cscript hello.wsf
```

This example assumes that you are running the script from the same directory it is located. Otherwise, the path to the file will need to be specified. Any arguments or options passed to the script would be added after the script name. So, if you wanted to use the Interactive mode options and pass a string as the first argument, your script call would look similar to this:

```
cscript hello.wsf //I /arg1
```

Command-based scripts can also display output, similar to Windows-based scripts. However, unlike the Windows-based pop-up window, the command script output is

suppressed to the same DOS Window in which the script is run. This is one of the disadvantages to the command-based scripts. No graphical components are provided. In Figure 6.5, an example of the well-known Hello World program is executed displaying the “Hello World” text to the screen.

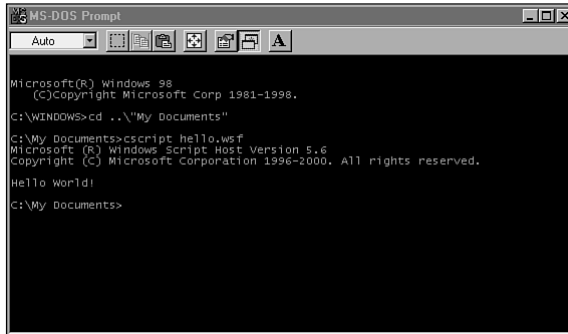


Figure 6.5

Command-based output display.

Remote Scripting

With the latest version of WSH (version 5.6), it is possible to execute scripts remotely. This allows you to remotely administer various automation tasks to computers across the network. These tasks can even be executed simultaneously. You can remotely start, stop, and get the status of WSH scripts. If an error occurs through the `WshRemoteError` object, you can get the character, line number, error description, and even source code for the error.

When a remote script is executed, the local machine actually copies the script to the remote machine before execution. Before a remote script can be executed though, the remote machine must be set up with the proper security settings. To set up remote scripting on your machine, you need to follow three simple steps. First install WSH, version 5.6 (if not already installed). The second step is to add yourself to the remote machine's local administrators group. The third step is to enable remote WSH through the `Poledit.exe` executable. After this is taken care of, your machine is ready for remote scripting!

There is a caveat to remote scripting. Remote scripting can only be run in the Windows 2000/NT environment. The Windows 9x scripting engine does not support this functionality. This is because you cannot authenticate users who are not on the NT domain.

How to Use Remote Scripting

So how do you use remote scripting? The WSH has provided the `WshRemote` object and various properties and methods to control the execution of your remote scripts. The `WshRemote` object is created through the `WshController` object. First create an instance

of the `WshController` with the `CreateObject` method. After you have the controller object, you can use the `CreateScript` method to specify which script you want to execute remotely. This method actually returns a `WshRemote` object that can be used with its `execute()` method to run the script remotely.

As with most computer programs, not all the scripts will run without errors. This is not a problem though. The WSH object model includes the `WshRemoteError` object, which can be used to obtain various error information. It is not an object that is directly instantiated, but an object that is available as a child object of the `WshRemote` object.

Remote Scripting Example

Lets take a look at an example of remote scripting. In Listing 6.5, a maintenance script is to be run remotely on a machine named "blue2". A Controller object, *WshCtrlObj*, is first created. Then an instance of the `WshRemote` object is created using the `CreateScript` method. The path of the script to be run and the machine name are passed as parameters. We set some events to be caught and execute the script. A print statement is added just to check the status. Our current script sleeps while waiting for the remote script to finish execution. Should any errors occur, the function `remote_error` can be used to display all the error information.

Listing 6.5 Remote Scripting Example

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\remote_tasks\maintenance.wsf",
➤ "blue2");

// Catch events.
WScript.ConnectObject(RemoteScript, "RemoteScript_");

// Start the script.
remoteScript.Execute();

// Print the current status.
WScript.Echo("The current status of the remote script is: " +
➤ remoteScript.Status);

while (remoteScript.Status !=2) {
    WScript.Sleep(100);
}

// Catch errors.
function remote_Error(){
```

```
var myError = remoteScript.Error;
var errorString = "An Error occurred in the remote execution\n";
errorString += "The remote error: " + myError.Description + "\n";
errorString += "occurred at line number: " + myError.Line + "\n";
errorString += " ,character: " + myError.Character + "\n";
errorString += "The Source for the Error is: " + myError.Source + "\n";
WScript.Echo(errorString);
}

</script>
</job>
```

Moving On

In this chapter, we have discussed many aspects of the Windows Script Host. We have taken a look at the object model and a few specific objects in detail. We discussed the two methods of running WSH scripts and even how to run scripts remotely. Although the WSH 5.6 is still in beta version, a lot of things can be done with it. Look for even more to be added as WSH continues to evolve.



PART III

JAVASCRIPT REFERENCE

7 Core Language 193

8 Client-Side 563

9 Server-Side 1175

CD-ROM

10 DOM Core **CD:1403**

11 DOM HTML **CD:1479**

12 DOM Events **CD:1731**

13 DOM CSS **CD:1767**

14 JScript RunTime **CD:1823**

15 Windows Script Host **CD:1927**



CHAPTER 7

Core Language

This chapter is a detailed reference of all the items and elements that make up the core JavaScript and JScript languages as well as information in the ECMA-262, 3rd Edition standard (ECMAScript). Because the details of JavaScript, JScript, and ECMAScript varies slightly, so does each one's definition of core. Because of these variations, this chapter covers items that are considered core in one standard, but might not exist in the other standards. Every nuance of the languages is covered, giving you a one-stop place to get answers to all your questions. Each entry includes version, environment support, syntax, a description, and an example, as well as many other details.

The chapter is in alphabetical order, by objects, to provide you with quick, easy access to the major parts of the language. The methods, properties, functions, and event handlers of every object appear alphabetically after the respective parent object, using the simple dot notation that makes the language so easy to learn. For example, the `sin()` function is found after the `Math` object as `Math.sin()`.

@_alpha

JScript3.0+

IE 4+

Syntax

@_alpha

Description

The @_alpha variable is used in JScript conditional compilation to determine if a DEC Alpha processor is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.1 alerts the user when a DEC Alpha processor is used.

Listing 7.1 *Detect a DEC Alpha Processor*

```
<script type="text/jscript" language="JScript">
<!--
@if (@_alpha)
    alert("You are using a DEC Alpha processor.");
@else
    alert("You are NOT using a DEC Alpha processor.");
@end

// -->
</script>
```

@_jscript

JScript3.0+

IE 4+

Syntax

@_jscript

Description

The @_jscript variable is used in JScript conditional compilation to determine if JScript is being used. This variable is always true.

Example

Listing 7.2 displays an alert box based on the value of the @_jscript variable.

Listing 7.2 *Detect a JScript*

```
<script type="text/jscript" language="JScript">
<!--
@if (@_jscript)
    alert("The @ jscript variable is true.");
@else
    alert("The @_jscript variable is NOT true.");
@end

// -->
</script>
```

@_jscript_build

JScript3.0+

IE 4+

Syntax

@_jscript_build

Description

The @_jscript_build variable is used in JScript conditional compilation to hold the build number of the JScript scripting engine.

CAUTION

Before accessing the @_jscript_build variable, use the @cc_on statement to define the variable. More information about the @cc_on statement can be found later in this chapter.

Example

Listing 7.3 uses the @_jscript_build variable to display the build number of the JScript engine that is being used.

Listing 7.3 Retrieve JScript Build Number

```
<script type="text/jscript" language="JScript">
<!--

//Set conditional compilation so @_jscript_build variable will be defined.
@cc_on

//Display the JScript build number using the @_jscript_build variable.
document.write("The JScript engine build number is ",@_jscript_build);

// -->
</script>
```

@_jscript_version

JScript3.0+

IE 4+

Syntax

@_jscript_version

Description

The @_jscript_version variable is used in the JScript conditional compilation to hold the JScript version number in *major.minor* format.

CAUTION

Before accessing the @_jscript_version variable, use the @cc_on statement to define the variable.

Example

Listing 7.4 uses the `@_jscript_version` variable to display the JScript version that is being used.

Listing 7.4 Determine JScript Version

```
<script type="text/jscript" language="JScript">
<!--

//Set conditional compilation so @_jscript_version variable will be defined.
@cc_on

//Display the JScript version number using the @_jscript_version variable
document.write("The JScript version is ",@_jscript_version).

// -->
</script>
```

@_mac

JScript3.0+

IE 4+

Syntax

`@_mac`

Description

The `@_mac` variable is used in JScript conditional compilation to determine if an Apple Macintosh system is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.5 alerts the user when an Apple Macintosh system is used.

Listing 7.5 Detect an Apple Macintosh System

```
<script type="text/jscript" language="JScript">
<!--

@if (@_mac)
    alert("You are using an Apple Macintosh system.");
@else
    alert("You are NOT using an Apple Macintosh system.");
@end

// -->
</script>
```

@_mc680x0

IE 4+

JScript3.0+

Syntax

@_mc680x0

Description

The @_mc680x0 variable is used in JScript conditional compilation to determine if a Motorola 680x0 processor is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.6 alerts the user when a Motorola 680x0 processor is used.

Listing 7.6 Detect a Motorola 680x0 Processor

```
<script type="text/javascript" language="JScript">
<!--

@if (@_mc680x0)
    alert("You are using a Motorola 680x0 processor.");
@else
    alert("You are NOT using a Motorola 680x0 processor.");
@end

// -->
</script>
```

@_PowerPC

IE 4+

JScript3.0+

Syntax

@_PowerPC

Description

The @_PowerPC variable is used in JScript conditional compilation to determine if a Motorola PowerPC processor is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.7 alerts the user when a Motorola PowerPC processor is used.

Listing 7.7 Detect a Motorola PowerPC Processor

```
<script type="text/jscript" language="JScript">
<!--

@if (@_PowerPC)
    alert("You are using a Motorola PowerPC processor.");
@else
    alert("You are NOT using a Motorola PowerPC processor.");
@end

// -->
</script>
```

@_win16

IE 4+
JScript3.0+

Syntax

@_win16

Description

The @_win16 variable is used in JScript conditional compilation to determine if a win16 system is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.8 alerts the user when a win16 system is used.

Listing 7.8 Detect a win16 System

```
<script type="text/jscript" language="JScript">
<!--

@if (@_win16)
    alert("You are using a 16 bit Windows system.");
@else
    alert("You are NOT using a 16 bit Windows system.");
@end

// -->
</script>
```

@_win32

JScript3.0+
IE 4+

Syntax

@_win32

Description

The @_win32 variable is used in JScript conditional compilation to determine if a win32 system is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.9 alerts the user when a win32 system is used.

Listing 7.9 Detect a win32 System

```
<script type="text/javascript" language="JScript">
<!--

@if (@_win32)
    alert("You are using a 32 bit Windows system.");
@else
    alert("You are NOT using a 32 bit Windows system.");
@end
// -->
</script>
```

@_x86

JScript3.0+
IE 4+

Syntax

@_x86

Description

The @_x86 variable is used in JScript conditional compilation to determine if a Intel processor is being used. When the variable is not true, it behaves as NaN although it is technically undefined.

Example

Listing 7.10 alerts the user when an Intel processor is used.

Listing 7.10 Detect an Intel Processor

```
<script type="text/javascript" language="JScript">
<!--

@if (@_x86)
    alert("You are using an Intel processor.");
```

Listing 7.10 Continued

```
@else
    alert("You are NOT using an Intel processor.");
@end

// -->
</script>
```

@cc_on

JScript3.0+
IE 4+

Syntax

```
@cc_on
```

Description

The @cc_on statement is used to activate JScript conditional compilation in the scripting engine.

Example

Listing 7.11 shows where the @cc_on statement would be used within JScript code to get the JScript version.

Listing 7.11 Activate Conditional Compilation

```
<script type="text/javascript" language="JScript">
<!--

//Set conditional compilation so @_jscript_version variable will be defined.
@cc_on

//Display the JScript version number using the @_jscript_version variable
document.write("The JScript version is ",@_jscript_version).

// -->
</script>
```

@if

JScript3.0+
IE 4+

Syntax

```
@if (condition1)
    statement1
@elif (condition2)
    statement2
```

```
@else
  statement3
@end
```

Description

The `@if` statement operates much the same as a typical `if` conditional statement, but it evaluates compiled variables and has a slightly different syntax. If the expression in *condition1* evaluates to true, *statement1* is executed. Otherwise, the *condition2* is evaluated to determine whether *statement2* should be evaluated. The `@elif` statement operates the same as the JavaScript `else...if` statement, even though it looks different. You can use as many `@elif` statements as needed or none at all. Only one `@else` statement should be used to catch conditions that don't meet any of the previous conditionals. Similar to the `@elif` statement, the `@else` statement is optional. The `@end` statement is required at the end of all `@if` conditionals.

Example

Listing 7.12 displays a message based on the type of processor being used.

Listing 7.12 Using `@if` to Display Processor Type

```
<script type="text/javascript" language="JScript">
<!--

//Alert the user as to which type of processor they are using.
@if(@_jscript)
  alert("You are using a DEC Alpha processor.");
@elif(@_mc680x0)
  alert("You are using a Motorola 680x0 processor.");
@elif(@_PowerPC)
  alert("You are using a Motorola PowerPC processor.");
@elif(@_x86)
  alert("You are using an Intel processor.");
@else
  alert("I don't know what type of processor you have!");
@end

// -->
</script>
```

@set

```
JScript3.0+
IE 4+
```

Syntax

```
@set @varName = value
```

Description

The @set property allows the creation of JScript variables. The name of the variable must start with an ampersand character. If the variable is used before being defined, its value will be NaN.

Example

Listing 7.13 uses the @set property to create a custom compiled variable called @number and assigns the number 25 to the variable. The variable is then used to display a sentence in the browser.

Listing 7.13 Create Custom Compiled Variable Using the @set Property

```
<script type="text/jscript" language="JScript">
<!--

//Create a custom compiled variable called @number and set it to 25
@set @number = 25.

//Display a sentence in the browser based on the value stored in the
//custom compiled variable @number.
@if(@number == 25)
    document.write("Number is equal to ",@number);
@else
    document.write("Number is not equal to 25.");
@end

// -->
</script>
```

Operators

- (Subtraction)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 - num2

Description

The number to the right of the operator is subtracted from the number to the left of the operator. If either operand is a string, an attempt is made to convert the string to a number before performing the operation.

Example

The code in Listing 7.14 creates a string out of the number "45". The string is converted to a number before being subtracted from the number 25. The result of the subtraction operation (20) is displayed.

Listing 7.14 Using the Subtraction Operator

```

<script type="text/javascript" language="JavaScript">
<!--

//Declare the number 45 as a string
aString = new String("45");

//Subtract 25 from 45
answer = aString - 25;
document.write("answer = (45-25)<br>");

//Answer is equal to 20
document.write("answer = ",answer);

// -->
</script>

```

- (Unary Negation)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

-num

Description

The unary negation operator changes the sign of *num*. When negating a variable, the contents of the variable do not change; only the value returned is negated. If the operand is a string, it is converted to a number before performing the unary negation operation.

Example

The code in Listing 7.15 creates a string "36". The string is converted to a number before being negated. The result of the negation is -36.

Listing 7.15 Using the Negation Operator

```

<script type="text/javascript" language="JavaScript">
<!--

//Declare the number 36 as a string
aString = new String("36");

//negate the number
answer = -aString;
document.write("answer = -aString<br>");

```

Listing 7.15 Continued

```
//Answer is equal to -36
document.write("answer = ",answer);

// -->
</script>
```

-- (Decrement)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
--variable
variable--
```

Description

The pre-decrement operator, defined in the first syntactical definition, decrements *variable* by 1. The new decremented value is returned by the operation.

The second syntactical definition contains the post-decrement operator. This operator is similar to the pre-decrement operator in that it decrements *variable* by 1. However, the original value is returned by the operation before being decremented.

In both cases, if the operand is a string, it is converted to a number before performing the operation.

Example

Listing 7.16 demonstrates how the pre-decrement and post-decrement operators work. Notice that the variable *num* holds a string that is converted before performing the decrement operations. The result from executing this code is shown in Figure 7.1.

Listing 7.16 Using the Decrement Operator

```
<script type="text/javascript" language="JavaScript">
<!--

// Store value in variable before pre-decrement
document.write("<h3>Before Pre-decrement</h3>");
num = new String("807"); //num holds the string 807
document.write("num=",num,"<br>"); //807 is displayed.

// Pre-decrement the value stored in num
returnValue = --num;
document.write("<h3>After Pre-decrement</h3>");
document.write("num=",num,"<br>"); //806 is displayed.

//806 is displayed by returnValue
document.write("Value returned from operation is ",returnValue,"<br>");
```

```
// Post-decrement the value stored in num
returnValue = num--;
document.write("<h3>After Post-decrement</h3>");
document.write("num=",num,"<br>");    //805 is displayed.

//806 is displayed by returnValue
document.write("Value returned from operation is ",returnValue,"<br>");

// -->
</script>
```

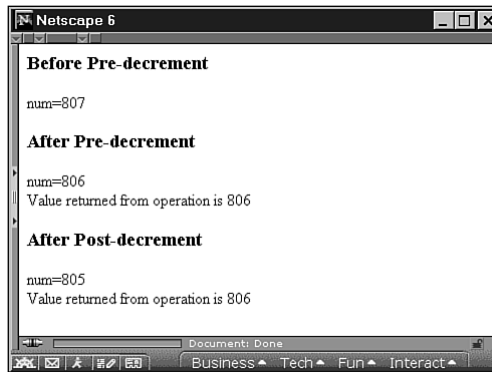


Figure 7.1

The difference between pre-decrement and post-decrement.

! (Logical NOT)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`!operand`

Description

The `!` operator inverts the boolean value returned from the *operand*. So, if *operand* evaluates to `true`, the result of the operation is `false`. If the *operand* evaluates to `false`, the result is `true`. When the *operand* evaluates to a non-Boolean value, it is converted to `true` or `false` before performing the inversion.

Example

In Listing 7.17 the character `"1"` is converted to the Boolean `true` before inverting the value with the NOT operand. The variable `theReverseTruth` is assigned `false`.

Listing 7.17 Using the Logical NOT Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a variable that contains 1
theTruth = new String("1");

//Invert the value stored in the previous variable
theReverseTruth = !theTruth;

//Output the values stored in the two variables
document.write("theTruth = ",theTruth,"<br>");
document.write("theReverseTruth = ",theReverseTruth);

// -->
</script>
```

!= (Not Equal)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

expression1 != *expression2*

Description

The not equal operator compares the first expression to the second expression. If the expressions are not equal, *true* is returned from the operation. If they are equal, *false* is returned.

JavaScript and Microsoft JScript interpreters attempt to convert the expressions to the same data type before evaluating the not equal operation using the following rules:

- *true* is converted to the number 1, and *false* is converted to zero before being compared.
- If either of the operands is NaN, the equality operator returns *false*.
- Null and undefined are equal.
- Null and undefined are not equal to 0 (zero), "" (empty string), or *false*.
- If a string and a number are compared, attempt to convert the string to a number and then check for equality.
- If an object and a string are compared, attempt to convert the object to a string and then check for equality.
- If an object and a number are compared, attempt to convert the object to a number and then check for equality.
- If both operands of an equality operation are objects, the address of the two objects are checked for equality.

CAUTION

In JavaScript 1.2, the decision was made to NOT do type conversion on the operands of the not equal operator. JavaScript reverted back to using type conversion with this operator in 1.3 and later.

Example

In the following example, the string "523" is converted to a number (except in JavaScript 1.2, in which type conversion is not performed), so that two numbers are compared. Because the left operand is equal to the right operand, the phrase "The string 523 is EQUAL to the number 523" is written to the browser window.

Listing 7.18 Using the Not Equal Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Before comparing this string to the number the
//string will be converted to a number.
if("523" != 523)
{
    //If the two numbers are not equal then display this sentence.
    document.write("The string 523 is NOT equal to the number 523");
}
else
{
    //If the two numbers are equal then display this sentence.
    document.write("The string 523 is EQUAL to the number 523");
}

// -->
</script>
```

!= (Non-Identity)

JavaScript 1.3+, ECMAScript 3E+, JScript 1.0+
Nav4.06+, IE 3+, Opera5+

Syntax

```
expression1 != expression2
```

Description

The non-identity operator compares the first expression to the second expression. If the value on the left is not equal to the value on the right side of the operator, true is returned from the operation. If the values are equal, false is returned.

NOTE

No type conversion is performed on the expressions before the comparison is made.

Example

In Listing 7.19, the string "8765" is NOT converted to a number, so the two expressions are not the same type. Because the two operands are not the same type, they are not equal, so the phrase "The string 8765 is NOT equal to the number 8765" is written to the browser window.

Listing 7.19 Using the Not-Identity Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Compare the string to the number but do not
//convert the string to a number.
if("8765" != 8765)
{
    //If the string and number are not equal then display this sentence.
    document.write("The string 8765 is NOT equal to the number 8765");
}
else
{
    //If the string and number are equal then display this sentence.
    document.write("The string 8765 is EQUAL to the number 8765");
}

// -->
</script>
```

% (Modulus)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 % num2

Description

The modulus operator begins the same as the division operator, by dividing the left value by the right; but instead of returning the normal result of division, only the remainder is returned by the operation. If either operand is a string, an attempt is made to convert the string to a number before performing the operation.

Example

In Listing 7.20, the variable `answer` is assigned the value 1 because 7 is divided by the number 2 three times plus a remainder of 1. This remainder is stored in the variable `answer`.

Listing 7.20 Using the Modulus Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//1 is stored in the variable answer.
answer = 7 % 2;

//Display the value in the variable answer
document.write("answer = ",answer);

// -->
</script>
```

%= (Modulus Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

`variable %= value`

Description

The modulus assignment operator divides the value stored in the left variable by the right value. The remainder is returned by the operation and stored in the variable to the left of the operator. If `value` is a string, an attempt is made to convert the string to a number before performing the modulus and assignment.

Example

In Listing 7.21, the variable `answer` is initially assigned the value 17. The number 17 is divided by the number 3. The remainder portion (modulos) of the result of the division is stored in the variable `answer`, overwriting the number 3.

Listing 7.21 Using the Modulus Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable answer with 3.
answer = 17;

//The number 2 is stored in the variable answer.
answer %= 3;
```



```
//Display the value stored in the variable answer.
document.write("11 & 6 = ",answer);

// -->
</script>
```

&& (Logical AND)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

expression1 && *expression2*

Description

The logical AND operator returns `true` if the expression on the left and the expression on the right of the operator evaluate to `true`. If either the left, right, or both evaluates to `false`, the result of the operation is `false`.

Be sure to note that the implementation of the logical AND operator in JavaScript is more complex than what was just mentioned. The AND operation begins by evaluating the left expression. If the left expression evaluates to `false`, the basic logic of the AND operator is complete, so the right expression is never evaluated. But if the left expression evaluates to `true`, the right expression must be evaluated to determine the final result of the AND operation. In either case, the final result returned by the AND operation is actually determined by the result of the last expression to be evaluated.

Example

Listing 7.23 demonstrates the complications associated with the logical AND operator. The first expression evaluates to `false`, causing the logical AND operation to evaluate to `false` and the message "The && evaluated FALSE!" is displayed. Because the first expression evaluates to `false`, the second expression, which assigns the number 5 to the variable `x`, is not evaluated and `x` remains equal to 3.

Listing 7.23 The Complicated Logical AND Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable x with the number 3.
x = 3;

//The assignment of 5 to the variable x never occurs
//since the expression (2==x) evaluates to false in the if statement.
if((2==x) && (x=5))
{
    document.write("The && evaluated TRUE!<br>");
}
```

&= (Bitwise AND Assignment)

Syntax

Description

Table 7.2 Bitwise AND Truth Table

Example

Listing 7.24 Using the Bitwise AND Plus Assignment Operator

```
// integer = 32-bit binary representation
// 12 = 00000000000000000000000000001100
```


Syntax

variable *= *value*

Description

The number stored in the variable to the left of the operator is multiplied by the number to the right of the operator. The result of the multiplication is written into the variable to the left of the operand. If either of the operands is a string, it is converted to a number.

Example

In Listing 7.26, the string "5" is converted to a number before being multiplied by the number 7. The result of the multiplication, which is 35, is stored in the variable x.

Listing 7.26 Using the Multiplication Plus Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable x with the number 7
//and the variable aString with "5".
x = 7;
aString = new String("5");

//Perform multiplication and assignment operation.

//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

, (Comma)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, IE 3+, Opera3+

Syntax

expression1, *expression2*, ..., *expressionN*

Description

The comma allows multiple expressions to be evaluated as one expression. The only value returned from this operation is the return value of the right-most expression.

Example

In Listing 7.27, the comma is used to execute multiple assignment statements on one line. The number 3 is stored in y, and then 9 is stored in z. Because z=9 is the last

statement in the group of comma-separated statements, its value is returned and stored in the variable `x`. The final values stored in each variable is displayed (`x=9, y=3, z=9`).

Listing 7.27 Using the Comma to Separate Multiple Statements

```
<script type="text/javascript" language="JavaScript">
<!--

//Assign values to all three variables.
x = (y = 3, z = 9);

//Display the values stored in all three variables.
document.write("x = ",x,"<br>y = ",y,"<br>z = ",z);

// -->
</script>
```

/ (Division)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`num1 / num2`

Description

The left number is divided by the right number. If either of the operands is a string, it is converted to a number.

Example

In Listing 7.28, the string "168" is converted to a number before being divided by the number 14. The result (12) of the division is stored in the variable `x` and displayed in the browser window.

Listing 7.28 Using the Division Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable aString with "168".
aString = new String("168");

//Create the variable x and set it equal to the number 12.
x = aString / 14;

//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

***/** / (Multi-line Comment)**

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, IE 3+, Opera3+

Syntax

```
/* comments */
```

Description

Every character that appears within the two comment tags is ignored by the JavaScript interpreter. Placing the tags on different lines allows comments to span multiple lines. Be careful not to nest comment tags within comment tags because this will lead to errors.

Example

Listing 7.29 demonstrates the use of multiple-line comments in JavaScript code.

Listing 7.29 Using the Multi-Line Comments

```
<script type="text/javascript" language="JavaScript">
<!--

document.write("<h2>Multi-line Comments</h2>");

/* Even though this sentences spans multiple lines it is treated
as a comment because it begins and ends with comment tags.
document.write("Not displayed!");
Notice that even the JavaScript statement above is treated as a comment*/

// -->
</script>
```

***//* (Comment)**

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, IE 3+, Opera3+

Syntax

```
// comment
```

Description

Every character that appears after this tag and on the same line as the tag is ignored by the JavaScript interpreter.

Example

Listing 7.30 demonstrates the use of the single line comment in JavaScript code.

Listing 7.30 Using Single Line Comments

```
<script type="text/javascript" language="JavaScript">
<!--

document.write("<h2>Single Line Comments</h2>");

//Everything on the following line is considered a comment.
//document.write("Not Displayed!");

// -->
</script>
```

/= (Division Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
variable /= value
```

Description

The number stored in the variable on the left of the operator is divided by the number on the right. The result of the division overwrites the value in the variable on the left of the operator. If either of the operands is a string, it is converted to a number.

Example

In Listing 7.31, the string "8" is converted to a number before being used to divide by the number 32. The result of the division, which is 4, is stored in the variable x.

Listing 7.31 Using the Division and Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variables x and aString.
x = 32;
aString = new String("8");

//The variable x is now equal to the number 4.
x /= aString;

//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

?: (Conditional)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

expression ? *v* *value1* : *value2*

Description

An expression that evaluates to a Boolean is always placed to the left of the question mark (?). If the expression evaluates to `true`, *value1* is returned from the operation. If the expression evaluates to `false`, *value2* is returned.

The same functionality of the conditional operator can be achieved with an `if...else` statement.

Example

In Listing 7.32, the conditional operator is shown along with a similar standard `if` statement. Both the conditional operator and the `if` statement have the same result, except that the conditional operator takes up less space. The string “The hamburgers are done!” are displayed for both operators.

Listing 7.32 The Conditional Operator and `if` Statement Are Compared

```
<script type="text/javascript" language="JavaScript">
<!--

// Set the cooking status flag to "YES".
doneCooking = "YES"

//Create 2 empty variables.
var message1;
var message2;

//Standard if statement
if (doneCooking == "YES")
    message1 = "The hamburgers are done!";
else
    message1 = "The hamburgers are still cooking.";

//Same statement using conditional operator
message2 = (doneCooking == "YES") ?
    "The hamburgers are done!" :
    "The hamburgers are still cooking.";

// Print the message to the screen. Notice both messages are the same!
document.write("The if statement returns: ",message1,"<br>");
document.write("The conditional operator returns: ",message2);
```


Listing 7.35 Continued

Display the number 13 in the browser window.

```
document.write("9 | 5 = ", (9 | 5) );
```

```
// -->
</script>
```

|| (Logical OR)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
expression1 || expression2
```

Description

The logical OR operator returns `true` if the left operand, right operand, or both operands evaluates to `true`. If both the operands evaluate to `false`, the result of the operation is `false`.

The implementation of the logical OR operator in JavaScript and JScript is more complex than what was just mentioned. The OR operation begins by evaluating the left operand. If the left operand evaluates to `true`, the basic logic of the OR operator is complete, so the right operand is never evaluated. But if the left operand evaluates to `false`, the right operand must be evaluated to determine the final result of the OR operation. In either case, the final result returned by the OR operation is actually the result of the last operand to be evaluated.

Example

Listing 7.36 demonstrates the complications associated with the logical OR operator. The first expression evaluates to `true`, causing the logical OR operation to evaluate to `true`. Because the first expression evaluates to `true`, the second expression, which assigns the number 7 to the variable `x`, is not evaluated so `x` remains equal to 8.

Listing 7.36 The Complicated Logical OR Operator

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Initialize the variable x with the number 8.
x = 8;
```

```
//The assignment of 7 to the variable x never occurs
//since the first expression (8==x) evaluates to true.
if((8==x) || (x=7))
{
    document.write("The || evaluated TRUE!<br>");
}
```

```

else
{
    document.write("The || evaluated FALSE!<br>");
}

// x is equal to 8.
document.write("x=",x,"<br>");

// -->
</script>

```

|= (Bitwise OR Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

variable |= *value*

Description

The bitwise OR assignment operator looks at the integer numbers on both sides of the operator as 32-bit binary numbers. The logical OR (|) operator individually evaluates each of the 32 bits representing the number on the left of the operator to the corresponding bit of the number on the right of the operator. The 32-bit binary result of the logical OR operation is converted to an integer value and stored in the variable on the left of the operator.

Example

Listing 7.37 uses the bitwise OR assignment operator on the numbers 2 and 5 to generate the number 7.

Listing 7.37 Using the Bitwise OR and Assignment Operator

```

<script type="text/javascript" language="JavaScript">
<!--

// integer = 32-bit binary representation
// 2 = 00000000000000000000000000000011
// 5 = 00000000000000000000000000000101
// 7 = 00000000000000000000000000000111

//Initialize the variable x with the number 2.
x = 2;

//Perform the bitwise OR assignment operation.
x |= 5;

```

Listing 7.37 Continued

```
//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

~ (Bitwise NOT)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

~operand

Description

The bitwise NOT operator begins by looking at the number to the right of the operator as a 32-bit binary number. Each bit of the given number is reversed so that all ones become zeros and all zeros become ones. The 32-bit binary result is converted to an integer value and returned from the bitwise NOT operation.

The result of inverting a number can be very confusing because of the way signed numbers are represented. Just remember that applying the bitwise NOT operator to a positive number will return the original number with the sign changed, minus one.

Example

The code in Listing 7.38 demonstrates the use of the bitwise NOT operator. Notice that the result of the operation is -3 , which is the original number (2) with the sign reversed (-2) minus 1.

Listing 7.38 The Complicated Bitwise NOT Operator

```
<script type="text/javascript" language="JavaScript">  
!!--  
  
// integer = 32-bit binary representation  
// 2 = 00000000000000000000000000000010  
// -3 = 1111111111111111111111111111101  
  
//Display the result of the Bitwise NOT Operator.  
document.write("-2 = ",(~2)); //Displays -3  
  
// -->  
</script>
```

+ (Addition)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

operand1 + operand2

Description

The addition operator provides two types of functionality depending on the data type of the operands. The first type of functionality is simple addition, in which the value on the left of the addition operator is added to the value on the right. For this type of addition, both operands must be numbers.

The second type of functionality provided by the addition operator is string concatenation. If either of the operands is a string, string concatenation is performed by first converting any non-string operand to a string. String concatenation is then performed by appending the string to the right of the operator to the end of the string located to the left of the operator.

Example

Listing 7.39 demonstrates the addition of numbers as well as string concatenation. Compare the Listing 7.39 to the result of executing the code as seen in Figure 7.2 to understand how addition is handled with various variable types.

Listing 7.39 *Addition of Numbers and String Concatenation*

```
<script type="text/javascript" language="JavaScript">
<!--

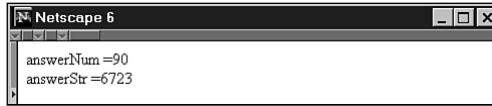
//Initialize the variable aString.
aString = new String("67");

//answerNum contains the number 90.
answerNum = 67 + 23;

//answerStr contains the string "6723".
answerStr = aString + 23;

//Print the result to the screen.
document.write("answerNum =",answerNum,"<br>");    //Displays 90
document.write("answerStr =",answerStr);            //Displays 6723

// -->
</script>
```

**Figure 7.2**

Using the addition operator to add numbers and concatenate strings.

++ (Increment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`++variable` (Pre-Increment)
`variable++` (Post-Increment)

Description

The pre-increment operator increments *variable* by 1. The new incremented value is returned by the operation.

The post-increment operator is similar to the pre-increment operator in that it increments *variable* by 1. However, the original value is returned by the operation before being incremented.

In both cases, if the operand is a string, it is converted to a number before performing the operation.

Example

Listing 7.40 demonstrates how the pre-increment and post-increment operators work. Notice that the variable *num* holds a string that is converted before performing the increment operation. The result of executing this code is shown in Figure 7.3.

Listing 7.40 *Using the Increment Operator*

```
<script type="text/javascript" language="JavaScript">
<!--

/* Store value in variable before pre-increment */
document.write("<h3>Before Pre-increment</h3>");
num = new String("807");    //num holds the string 807
document.write("num=",num,"<br>");    //807 is displayed.

/* Pre-increment the value stored in num */
returnValue = ++num;
document.write("<h3>After Pre-increment</h3>");
document.write("num=",num,"<br>");    //808 is displayed.
```



```
//808 is displayed by returnValue.
document.write("Value returned from operation is ",returnValue,"<br>");

/* Post-increment the value stored in num */
returnValue = num++;
document.write("<h3>After Post-increment</h3>");
document.write("num=",num,"<br>");    //809 is displayed.

//808 is displayed by returnValue.
document.write("Value returned from operation is ",returnValue,"<br>");

// -->
</script>
```

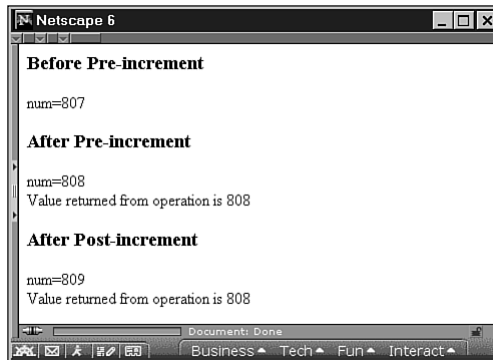


Figure 7.3

The difference between pre-increment and post-increment.

+= (Addition Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

variable += value

Description

The addition assignment operator provides two types of functionality depending on the data type of the operands. The first type of functionality is simple addition in which the value stored in the variable on the right of the addition assignment operator is added to the value on the left. The result of the addition overwrites the value stored in the variable on the left of the operator. If either of the operands is not a number or string, it will be converted to a number.

The second type of functionality provided by the addition assignment operator is string concatenation. If either of the operands is a string, string concatenation is performed by first converting any non-string operand to a string. String concatenation is then performed by appending the string to the right of the operator to the end of the string located to the left of the operator. The new string is stored in the variable on the left of the operator.

Example

In Listing 7.41, the addition assignment operator is used to add two numbers together as well as concatenate two strings.

Listing 7.41 Using the Addition and Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variables num and str.
num = 42;
str = new String("42");

//Use the addition/assignment operator.
num += 8;    //num contains the number 50.
str += 8;    //str contains the string "428".

//Display the results of the addition/assignment operator.
document.write("num = ",num,"<br>str = ",str);

// -->
</script>
```

< (Less Than)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 < *num2*

Description

The less than operator compares the value on the left of the operator to the value on the right. If the value on the left is less than the value on the right, `true` is returned from the operation. If the value on the left of the operator is greater than or equal to the value on the right, `false` is returned. If either of the operands is not a number or string, it will be converted to a number before performing the comparison.

Example

In Listing 7.42, the string "45" is converted to a number before performing the comparison. Because the number 45 is less than the number 68, the phrase "45 is less than 68" is returned.

Listing 7.42 Using the Less Than Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable str with "45".
str = new String("45");

//Compare the variable to the number 68.
if(str < 68)
    document.write("45 is less than 68");
else
    document.write("Returned FALSE!");

// -->
</script>
```

<< (Shift Left)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
num1 << num2
```

Description

The shift left operator looks at the integer on the left of the operator as a 32-bit binary number. The number of positions specified by *num2* shifts all the bits of *num1* to the left. As the bits are shifted to the left, zeros are filled in on the right. Because the number can only be 32-bits long, the extra bits on the left are lost. The 32-bit binary result of the shifting operation is converted to an integer value and returned from the shift left operation.

NOTE

The result generated from the shift left operator can be quickly calculated by multiplying the number by 2 raised to the *x* power, where *x* is the number of positions shifted.


```
// integer = 32-bit binary representation
// 3 = 00000000000000000000000000000011
// 12 = 00000000000000000000000000001100

//Initialize the variable x with the number 3.
x = 3;

//The variable x is now equal to the number 12.
x <= 12;

//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

<= (Less Than or Equal)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 <= *num2*

Description

The less than or equal operator compares the number on the left of the operator to the number on the right. If the number on the left is less than or equal to the number on the right, true is returned from the operation. If the number on the left of the operator is greater than the number on the right, false is returned.

Example

In Listing 7.45, the string "34" would be converted to a number before performing the comparison. Because the number 34 is less than the number 77, the phrase "34 is less than or equal to 77" would be returned.

Listing 7.45 Using the Less Than or Equal Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable str to "34".
str = new String("34");

//Compare the value stored in the variable to the number 77.
if(str <= 77)
    document.write("34 is less than or equal to 77");
```

Listing 7.45 Continued

```
else
    document.write("Returned FALSE!");

// -->
</script>
```

= (Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

variable = *value*

Description

The value to the right of the operator is stored in the variable to the left of the operator.

Example

In Listing 7.46, the assignment operator is used to assign various types of values to variables. The type of operator is then used to determine the type value stored in the variables. The types are then displayed in the browser (*x* is a number, *y* is a string, and *z* is a boolean).

Listing 7.46 Using the Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Number
x = 456;                //x contains a number.
document.write("x is a ",typeof x,"<br>");

//String
y = new String("Hello") //y contains a String.
document.write("y is a ",typeof y,"<br>");

//Boolean
z = true;               //z contains a Boolean.
document.write("z is a ",typeof z);

// -->
</script>
```

-= (Subtraction Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

variable -= value

Description

The number to the right of the operator is subtracted from the number stored in the variable to the left of the operator. The result of the operation overwrites the value stored in the variable to the left of the operator. If either operand is a string, an attempt is made to convert the string to a number before performing the subtraction.

Example

In Listing 7.47, the string "878" is converted to a number before the subtraction operation begins. The number 55 is subtracted from 878 and the result, 823, is stored in the variable `answer`.

Listing 7.47 Using the Subtraction Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable aString with "878".
aString = new String("878");

//Assign the number 823 to the variable answer.
answer -= 55;

//Display the values stored in the variable answer.
document.write("answer = ",answer);

// -->
</script>
```

== (Equal)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

expression1 == expression2

Description

The equal operator compares the value on the left of the operator to the value on the right of the operator. If the values are equal, `true` is returned from the operation. If the values are not equal, `false` is returned from the operation.

JavaScript attempts to convert the operands to the same data type before comparing the values for all versions of JavaScript except 1.2. JavaScript adheres to the following rules when performing type conversion:

- True is converted to the number 1, and false is converted to 0 before being compared.
- If either of the operands is NaN, the equality operator returns false.
- Null and undefined are equal.
- Null and undefined are not equal to 0 (zero), "" (empty string), or false.
- If a string and a number are compared, attempts to convert the string to a number and then checks for equality.
- If an object and a string are compared, attempts to convert the object to a string and then checks for equality.
- If an object and a number are compared, attempts to convert the object to a number and then checks for equality.
- If both operands of an equality operation are objects, the address of the two objects are checked for equality.



CAUTION

In JavaScript 1.2, the decision was made to NOT do type conversion on the operands of the equal operator. JavaScript reverted back to using type conversion with this operator in 1.3 and later.

Example

In Listing 7.48, the string "749" is converted to a number (except in JavaScript 1.2, in which type conversion is not performed) so that two numbers are compared. Because the left operand is equal to the right operand, the phrase "The string 749 is EQUAL to the number 749" is written to the browser window.

Listing 7.48 Using the Equal Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//The string is converted to a number before performing the comparison.
if("749" == 749)
{
    document.write("The string 749 is EQUAL to the number 749");
}
else
{
    document.write("The string 749 is NOT equal to the number 749");
}

// -->
</script>
```

=== (Identity)

JavaScript 1.3+, ECMAScript 3E+, JScript 1.0+
Nav4.06+, IE 3+, Opera5+

Syntax

expression1 === *expression2*

Description

The identity operator compares the first operand to the second operand. If the value on the left is equal to the value on the right side of the operator, `true` is returned from operation. If the values are not equal, `false` is returned.

NOTE

No type conversion is performed on the operands before the comparison is made.

Example

In Listing 7.49, the string "326" is NOT converted to a number, so the two operands are not the same type. Because the two operands are not the same type, they are not equal, so the phrase "The string 326 is NOT equal to the number 326" is written to the browser window.

Listing 7.49 Using the Identity Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//A string is compared to a number.
if("326" === 326)
{
    document.write("The string 326 is NOT equal to the number 326");
}
else
{
    document.write("The string 326 is EQUAL to the number 326");
}
// -->
</script>
```

> (Greater Than)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 > *num2*

Description

The greater than operator compares the value on the left of the operator to the value on the right. If the value on the left is greater than the value on the right, `true` is returned from the operation. If the value on the left of the operator is less than or equal to the

value on the right, `false` is returned. If either of the operands is not a number, it is converted to a number before performing the comparison.

Example

In Listing 7.50, the string `"112"` would be converted to a number before performing the comparison. Because the number 112 is greater than the number 68, the phrase `"112 is greater than 68"` is returned.

Listing 7.50 Using the Greater Than Operator

```
<script type="text/javascript" language="JavaScript">
<!--

//Initialize the variable str with the string "112".
str = new String("112");

//Compare the value in the variable to a number.
if(str > 68)
    document.write("112 is greater than 68");
else
    document.write("Returned FALSE!");

// -->
</script>
```

>= (Greater Than or Equal)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+ IE 3+, Opera3+

Syntax

`variable >= value`

Description

The greater than or equal operator compares the number on the left of the operator to the number on the right. If the number on the left is greater than or equal to the number on the right, `true` is returned from the operation. If the number on the left of the operator is less than the number on the right, `false` is returned.

Example

In Listing 7.51, the string `"95"` would be converted to a number before performing the comparison. Because the number 95 is greater than the number 44, the phrase `"95 is greater than or equal to 44"` is returned.

Listing 7.51 Using the Greater Than Or Equal Operator

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Initialize the variable str.
str = new String("95");

//Compare the value stored in the variable to the number 44.
if(str >= 44)
    document.write("95 is greater than or equal to 44");
else
    document.write("Returned FALSE!");

// -->
</script>
```

>> (Shift Right with Sign)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

num1 >> *num2*

Description

The shift right with sign operator looks at the integer to the left of the operator, *num1*, as a 32-bit binary number. All the bits in this number are shifted to the right by the number of positions specified by *num2*. As the bits are shifted to the right, either ones or zeros are filled in on the left. If the original number is positive, zeros are added to the left side of the binary number. On the other hand, if the original number is negative, ones are used. Because the result can only be 32-bits long, the extra bits on the right are lost. The 32-bit binary result of shifting operation is converted to an integer value and returned from the shift right with sign operation.

NOTE

The result generated from the shift right with sign operator can be quickly calculated by dividing the number by 2 raised to the *x* power, where *x* is the number of positions shifted. Discard the remainder.

Example

Listing 7.52 shifts the bits that make up the number 14 to the right two positions, which results in the number 3. The code also shifts the bits that make up the number -14 to the right one position, which results in the number -2.

Listing 7.52 Using the Shift Right with Sign Operator

```
<script type="text/javascript" language="JavaScript">
<!--
```

Listing 7.52 Continued

```
// integer = 32-bit binary representation
// 14 = 0000000000000000000000000000000000000000001110
// 3 = 0000000000000000000000000000000000000000000011

//The number 3 is assigned to the variable x.
x = 14 >> 2;

//Display the value stored in the variable.
document.write("14 >> 2 = ",x);

//-----
document.write("<br>");

// integer = 32-bit binary representation
// -4 = 111111111111111111111111111111111111111111100
// -2 = 11111111111111111111111111111111111111111110

//The number -2 is stored in the variable y.
y = -4 >> 1; //y is equal to -2.

//Display the value stored in the variable y.
document.write("-4 >> 1 = ",y);

// -->
</script>
```

>>= (Shift Right with Sign Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
variable >= v value
```

Description

The shift right with sign assignment operator (`>>=`) looks at the integer to the left of the operator as a 32-bit binary number. All the bits in this number are shifted to the right by the number of positions specified by the integer to the right of the operator. As the bits are shifted to the right, either ones or zeros are filled in on the left. If the original number is positive, zeros are added to the left side of the binary number. On the other hand, if the original number is negative, ones are used. Because the result can only be 32-bits long, the extra bits on the right are lost. The 32-bit binary result of shifting operation is converted to an integer value and stored in the variable to the left of the operator.

Listing 7.54 Using the Shift Right Zero Fill Operator

```
<script type="text/javascript" language="JavaScript">  

```

>>>= (Shift Right Zero Fill Assignment)

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+

Syntax

```
variable >>= value
```

Description

The shift right zero fill assignment operator (`>>=`) looks at the integer to the left of the operator as a 32-bit binary number. All the bits in this number are shifted to the right by the number of positions specified by the integer to the right of the operator. As the bits are shifted to the right, zeros are filled in on the left, regardless of the sign of the original integer. Because the result can only be 32-bits long, the extra bits on the right are lost. The 32-bit binary result of this shifting operation is converted to an integer value and stored in the variable to the left of the operator.

Example

In Listing 7.55, the 32-bit binary version of the number -6, which is stored in the variable `x`, is shifted one position to the right. The result of this operation, the number 1073741822, is stored in the variable `x`.

Listing 7.55 Using the Shift Right Zero Fill Assignment Operator

```
<script type="text/javascript" language="JavaScript">
<!--

// integer = 32-bit binary representation
//      -6 = 111111111111111111111111111111010
// 1073741822 = 00111111111111111111111111111110
//Initialize the variable x with the number -6.
x = -6;

//The variable x now contains the number 1073741822.
x >>= 2;

//Display the value stored in the variable x.
document.write("x = ",x);

// -->
</script>
```

abstract

JavaScript 1.2+, ECMAScript 2E+, JScript 5+
Nav4+, NES 3+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The `abstract` keyword has not been implemented in JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented, therefore no example is provided.

ActiveXObject

JScript3.0+
IE4+

Syntax

```
var variable = new ActiveXObject(serverName.typeName,location)
```

Description

The `ActiveXObject` creates a reference to an object that is connected to another application or programming tool through automation interfaces. The arguments used by this object are listed in Table 7.6.

Table 7.6 Arguments Associated with `ActiveXObject`

Argument	Description
serverName	The name of the application that provides the object.
typeName	The type or class of the object to create.
location	The name of the network server where the object is to be created. This argument is optional.

Example

Listing 7.56 uses the `ActiveXObject` to create a multiplication table in a Microsoft Excel document. Excel will be started automatically.

Listing 7.56 Create a Multiplication Table in an Excel Document

```
<script type="text/jscript" language="JScript">
<!--

var ExcelSheet = new ActiveXObject("Excel.Sheet");

// Make Excel visible through the Application object.
ExcelSheet.Application.Visible = true;

//Create multiplication table in Excel.
for(i=1;i<11;i++)
{
    // numbers to be multiplied in first two rows.
    ExcelSheet.ActiveSheet.Cells(i,1).Value = i;
    ExcelSheet.ActiveSheet.Cells(i,2).Value = 9;

    // Create Excel string to handle multiplication.
    var aString = new String("=A");
    aString += i;
    aString += "*B";
    aString += i;
    ExcelSheet.ActiveSheet.Cells(i,3).Value = aString;
}

// -->
</script>
```


Arguments

JavaScript 1.1+, ECMAScript 1E+, JScript 1.0+
Nav3+, IE3+, Opera5+

Syntax

arguments
arguments [*index*]

Description

The Arguments object is an array that contains all the arguments passed into the currently executing function as well as a few other useful properties. This object is automatically created and initialized when a function is invoked and goes out of scope as soon as the code function finishes executing. To access arguments passed into a function, simply use array brackets to specify an *index*. Table 7.7 lists the properties associated with the Argument object.

NOTE

To use the Arguments object, you do not specify the function using dot notation as you might expect. This is because this object is different from the Function. arguments[] array associated with a Function object. Using this Arguments object gives you the ability to access the arguments of functions that have no name.

Table 7.7 Properties Associated with the Arguments Object

Property	Description
callee	Contains the function that is currently executing
caller	Contains the Arguments object of the calling function
length	The length of the arguments array

Example

In Listing 7.57, a function is created to display an individual's favorite food in an alert box. The function is called when the buttons are pressed. The arguments passed to this function are accessed within the function with the Arguments object array brackets.

Listing 7.57 Using the Arguments Object to Display People's Favorite Foods

```
<html>
<body>
Select a person's name to discover their favorite food!

<form>
<input type="button"
value="Meredith"
OnClick=displayFood(this,"pizza")>
```

Listing 7.57 Continued

```
<input type="button"
      value="Allison"
      OnClick=displayFood(this,"beans")>
<input type="button"
      value="BayLeigh"
      OnClick=displayFood(this,"carrots")>
<input type="button"
      value="Michael"
      OnClick=displayFood(this,"corn")>
<input type="button"
      value="Rob"
      OnClick=displayFood(this,"hotdogs")>
</form>

<script type="text/javascript" language="JavaScript">
<!--

//Create a function that displays a person's favorite food.
function displayFood()
{
    //Create a string that contains the name of the person.
    var aString = arguments[0].value;
    aString += "'s favorite food is ";

    //Add the favorite food to the end of the string.
    aString += arguments[1];

    //Display the string in an alert box.
    alert(aString);
}

// -->
</script>

</body>
</html>
```

Arguments.callee

JavaScript 1.2+, ECMAScript 1E+, JScript 5.5+
Nav4+, IE5.5+, Opera5+

Syntax

arguments.callee

Description

The callee property of the Arguments object contains the function that is currently executing. This is useful if the function has no name.

Example

In Listing 7.58, a function is created to display an individual's favorite food in an alert box. The function is called when the buttons are clicked. The code that makes up the executing function is displayed using the callee property.

Listing 7.58 Using the callee *Property of the* Arguments *Object*

```
<html>
Learn what functions are used when you press the buttons
below that represent a person's favorite food!

<form>
<input type="button"
      value="Meredith"
      OnClick=displayFood(this,"pizza")>
<input type="button"
      value="Allison"
      OnClick=displayFood(this,"beans")>
<input type="button"
      value="BayLeigh"
      OnClick=displayFood(this,"carrots")>
<input type="button"
      value="Michael"
      OnClick=displayFood(this,"corn")>
<input type="button"
      value="Rob"
      OnClick=displayFood(this,"hotdogs")>
</form>

<script type="text/javascript" language="JavaScript">
<!--

//Create a function that displays a person's favorite food.
function displayFood()
{
    //Create a string that contains the name of the person.
    var aString = arguments[0].value;
    aString += "'s favorite food is ";

    //Add the favorite food to the end of the string.
    aString += arguments[1];

    //Display the string in an alert box.
    alert(aString);

    //Display the function using the callee property.
    alert(arguments.callee.toString());
}
```

Listing 7.58 Continued

```
// -->
</script>
</html>
```

Arguments.caller

JavaScript 1.2+
Nav4+

Syntax

arguments.caller

Description

The caller property of the Arguments object contains the Arguments object of the calling function. If the given function was not executed from within another function, null is stored in this property.

Example

Listing 7.59 creates two functions. One displays an individual's favorite food in an alert box, whereas the other displays the number of arguments associated with the calling function. Anytime a button is clicked, an alert box displays the favorite food. This function then calls the second function to display an alert box saying two arguments were passed into the first function.

Listing 7.59 Using the caller Property of the Arguments Object

```
<html>
Select a person's name to discover their favorite food!

<form>
<input type="button"
       value="Meredith"
       OnClick=displayFood(this,"pizza")>
<input type="button"
       value="Allison"
       OnClick=displayFood(this,"beans")>
<input type="button"
       value="BayLeigh"
       OnClick=displayFood(this,"carrots")>
<input type="button"
       value="Michael"
       OnClick=displayFood(this,"corn")>
<input type="button"
       value="Rob"
       OnClick=displayFood(this,"hotdogs")>
</form>
```

```
<script type="text/javascript" language="JavaScript">
<!--

//Display the number of arguments in the function that calls this function.
function displayArgLength()
{
    var argLengthStr = "The calling function contained ";
    argLengthStr += arguments.caller.length;
    argLengthStr += " arguments.";
    alert(argLengthStr);
}

//Create a function that displays a person's favorite food.
function displayFood()
{
    //Create a string that contains the name of the person.
    var aString = arguments[0].value;
    aString += "'s favorite food is ";

    //Add the favorite food to the end of the string.
    aString += arguments[1];

    //Display the string in an alert box.
    alert(aString);

    displayArgLength();
}

// -->
</script>
</html>
```

Arguments.length

JavaScript 1.1+, ECMAScript 1E+, JScript 5.5+
Nav3+, IE5.5+, Opera5+

Syntax

arguments.length

Description

The length property of the Arguments object contains the number of arguments that were passed into the function to which the Arguments object is associated. If fewer arguments are passed in than are specified in the definition of the function, the length property will only contain the number of arguments passed into the function. This number matches the number of elements in the arguments array associated with the Arguments object.

Example

In Listing 7.60, the length property of the Arguments object is used to process any number of arguments passed into the displayFood() function.

Listing 7.60 *Using the length Property to Access Elements of the Arguments Object*

```
<html>
Select a person's name to discover their favorite foods!

<form>
<input type="button"
      value="Meredith"
      OnClick=displayFood(this,"pizza","salad","cake")>
<input type="button"
      value="Allison"
      OnClick=displayFood(this,"beans","potatoes")>
<input type="button"
      value="BayLeigh"
      OnClick=displayFood(this,"carrots")>
<input type="button"
      value="Michael"
      OnClick=displayFood(this,"corn","beans")>
<input type="button"
      value="Rob"
      OnClick=displayFood(this,"hotdogs")>
</form>

<script type="text/javascript" language="JavaScript">
<!--

//Create a function that displays a person's favorite foods.
function displayFood()
{
    //Create a string that contains the name of the person.
    var aString = arguments[0].value;
    aString += "'s favorite foods are: ";

    //Add all the favorite foods to the end of the string.
    for(var i=1; i<arguments.length; i++)
    {
        aString += arguments[i];
        aString += ", ";
    }

    //Display the string in an alert box.
    alert(aString);
}
```

```
// -->  
</script>
```

Array

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

```
var variable = new Array()  
var variable = new Array(int)  
var variable = new Array(arg1, ..., argN)
```

Description

Although arrays can be created with the basic JavaScript object, the Array object provides a much easier way to create and manage arrays.

Table 7.8 lists the argument and return values associated with this object's constructors. These constructors create a new array and, in two cases, initialize the Array object based on the arguments passed in the parameter list. The constructor that has no arguments sets the length property to 0.

Table 7.8 *Arguments and Return Values Associated with the Array Object*

Type	Item	Description
Arguments	int	When the array constructor contains one argument, an array is created, and its length property is set to the value int.
	arg1,...argN	When the parameter list of the array constructor contains more than one argument, an array is created and the array is populated with the arguments. The array length property is set to the number of arguments in the parameter list.
Returns		The newly created array is returned from the constructor.

Table 7.9 lists the properties and methods used by the Array object.

Table 7.9 *Properties and Methods Used by the Array Object*

Type	Item	Description
Property	constructor	Specifies the function that creates the Array object's prototype.
	index	Contains the original string against which a regular expression was matched (read-only).

Table 7.9 Continued

Type	Item	Description
Methods	input	Contains the position of a regular expression match in a string (read-only).
	lastIndex	Contains the position after the last regular expression match in a string (read-only).
	length	Contains the number of elements in the array.
	concat()	Concatenates an array onto the end of an array.
	join()	Concatenates all elements of an array into one string.
	pop()	Deletes the last element from an array.
	push()	Adds elements to the end of an array.
	reverse()	Reverses the order of the elements in the array.
	shift()	Deletes elements from the front of an array.
	slice()	Returns a subsection of the array.
	sort()	Sorts elements in array.
	splice()	Inserts and removes elements from an array.
	toSource()	Converts elements to a string with square brackets.
	toString()	Converts elements to a string.
	unshift()	Addselements to the front of an array.
	unwatch()	Removes a watchpoint.
	watch()	Sets a watchpoint.

Example

Listing 7.61 creates an array of numbers using the Array constructor. Once created, the elements are displayed on the screen using bracket notation ([]).

Listing 7.61 Creating an Array and Accessing Its Elements

```
<html>

<h2>Creating and Accessing Arrays</h2>

<script type="text/javascript" language="JavaScript">
<!--

//Create a new array that contains 3 numbers.
numArray = new Array(45,67,34);
document.write("Created an array of numbers that contains 45, 67, and 34<br>");

//Display the contents of the array.
document.write("[0]=",numArray[0],"<br>"); // will display 45
document.write("[1]=",numArray[1],"<br>"); // will display 67
document.write("[2]=",numArray[2]);        // will display 34
```



```
// -->
</script>

</html>
```

Array.concat()

JavaScript 1.2+, ECMAScript 3E+, JScript 3.0+
Nav4+, NES3+, IE 4+, Opera5+

Syntax

```
array.concat (arg1,...argN)
```

Description

The `concat()` method adds the elements listed in the parameter list to the end of the existing array and returns the result. The original is not changed by this method. Should any of the arguments be Array, the elements of that array are concatenated to the array that called the method.

Table 7.10 lists the argument and return values associated with this method.

Table 7.10 Arguments and Return Values Associated with `concat()`

Type	Item	Description
Arguments	<code>arg1,...argN</code>	The parameter list of the <code>concat()</code> method contains one or more elements to be concatenated to the end of the array.
Returns		The original array with the new concatenated elements is returned from the method.

Example

Listing 7.62 uses the `concat()` method to display the total inventory of two grocery store shelves. Notice how the multidimensional arrays were concatenated together in Figure 7.4.

Listing 7.62 Using `concat()` to Display Inventory

```
<script type="text/javascript" language="JavaScript">
<!--
//Display the elements in the array.
function displayElements(theArray)
{
    //Access each element in the array.
    for(i=0; i<theArray.length; i++)
    {
        //Display the element.
        document.write("  - ",theArray[i][1]," ");
    }
}
```

Listing 7.62 Continued

```
        document.write(theArray[i][0], "<br>");
    }
}

//Create a grocery shelf using an array to represent
//the items on each shelf.
shelf1 = new Array(["apples",10],["oranges",25]);
document.write("Shelf 1 contains:<br>");

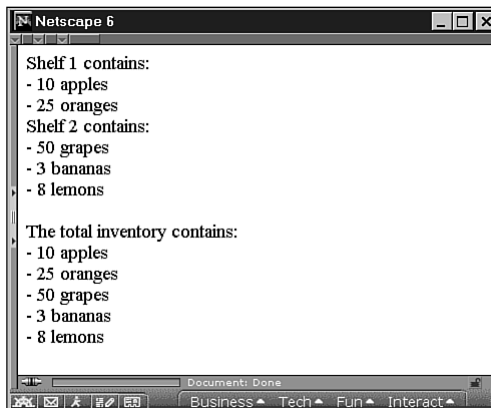
//Display the items on shelf 1.
displayElements(shelf1);

//Create a second grocery shelf using an array to represent
//the items on each shelf.
shelf2 = new Array(["grapes",50],["bananas",3],["lemons",8]);
document.write("Shelf 2 contains:<br>");

//Display the items on shelf 2.
displayElements(shelf2);

//Create a master inventory list by concatenating
//the two shelf arrays into one array.
inventory = shelf1.concat(shelf2);

//Display all the items on all the shelves.
document.write("<br>The total inventory contains:<br>");
displayElements(inventory);
// -->
</script>
```

**Figure 7.4**

Concatenate the items on two grocery store shelves.

Array.constructor

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

`array.constructor`

Description

The constructor property of the Array object is used to create the object's prototype.

Example

Listing 7.63 demonstrates the constructor property as it relates to the Array object.

Listing 7.63 The Array's constructor Property

```
<script type="text/javascript" language="JavaScript">
<!--
// Create a new array object using the constructor property.
myArray = new Array("Mike", "Eric", "Ory", "Jason", "Jonathon", "Kevin");
if(myArray.constructor == Array){
    document.write("Object was created");
}
// -->
</script>
```

Array.index

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE 4+, Opera4+

Syntax

`array.index`

Description

The index property is a read-only property that contains the position of a regular expression match within a string. This property only appears in the Array object that is returned from the following regular expression related methods:

- `RegExp.exec()`
- `String.match()`
- `String.replace()`

Example

Listing 7.64 creates a regular expression string that contains the search string "red". The `exec()` method is used to find the regular expression within a string. The `index` property is then used to display "The word 'red' was found at position 4", which contains the position character position (4) within the string where the word "red" was found.

Listing 7.64 Using the `index` Property to Display the Position of a Match

```
<script type="text/javascript" language="JavaScript">
<!--
myReg=/red/;
myArray = myReg.exec("The red ball.");
document.write("The word 'red' was found at position ",myArray.index);
// -->
</script>
```

Array.input

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE 4+, Opera4+

Syntax

`array.input`

Description

The `input` property is a read-only property that contains the string against which the regular expression was matched. This property only appears in the `Array` object that is returned from the following regular expression related methods:

- `RegExp.exec()`
- `String.match()`
- `String.replace()`

Example

Listing 7.65 creates a regular expression string that contains the search string "red". The `exec()` method is used to find the regular expression within a string. The `input` property is then used to display the string that was searched.

Listing 7.65 Using the `index` Property to Display the Regular Expression

```
<script type="text/javascript" language="JavaScript">
<!--
myReg=/red/;
myArray = myReg.exec("The red ball.");
document.write("The word red was found in the string '", myArray.input, "'");
// -->
</script>
```

Array.join()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

`array.join()`
`array.join(string)`

Description

The `join()` method converts all the elements of the array to strings and then concatenates all the strings into one string. If an argument is provided in the parameter list, it is used to separate the elements in the string returned by the method. Table 7.11 lists the argument and return values associated with this method.

Table 7.11 Arguments and Return Values Associated with `join()`

Type	Item	Description
Arguments	string	A string that is used to separate the elements of the array in the string returned from the method.
Returns		All the elements of the array are concatenated into one string and returned from the method.

Example

Listing 7.66 creates an array of fruits. The contents of this array are displayed on the screen using the `join()` method. A dash is specified as the delimiter used to separate the array elements when they are written to the screen as a string.

Listing 7.66 Using the `join()` Method to Display the Elements of an Array

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array that contains 3 fruits.
fruit = new Array("Apple","Orange","Grape");

//List the contents of the fruit array as a string
//with a dash separating each item.
aString = fruit.join("-"); //aString = "Apple-Orange-Grape"

//Display the string generated from using the join() method.
document.write("The fruit array contains: ",aString);

// -->
</script>
```

Array.lastIndex

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE 4+

Syntax

`array.lastIndex`

Description

The `lastIndex` property is a read-only property that contains the position following the last character in a regular expression match. This property only appears in the `Array` object that is returned from the following regular expression related methods:

- `RegExp.exec()`
- `String.match()`
- `String.replace()`

Example

Listing 7.67 creates a regular expression string that contains the search string "red". The `exec()` method is used to find the regular expression within a string. The `lastIndex` property is then used to display the number 7, which is the position following the word "red" that was found in the string.

Listing 7.67 Using the `lastIndex` Property to Display the Position Following the Match

```
<script type="text/javascript" language="JavaScript">
<!--
myReg=/red/;
myArray = myReg.exec("The red ball.");
document.write("The lastIndex property is set to ",myArray.lastIndex);
// -->
</script>
```

Array.length

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

`array.length`

Description

The `length` property holds the number of elements in the array. This property is a read/write variable. If the `length` property is overwritten with a number that is larger than the original number, new elements are added to the end of the array and assigned undefined values. If the `length` property is overwritten with a number that is smaller than the original number, elements at the end of the array are lost.

Example

Listing 7.68 creates an array of coins. The number of coins in the array is then reduced from 4 to 3 by modifying the `length` property so the array contains only three elements. Because the quarter was the last element in the array, it was removed when the `length` property was changed.

Listing 7.68 Using the `length` Property to Reduce the Number of Elements in an Array

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array of coins.
coins = new Array("Penny","Nickel","Dime","Quarter");

x = coins.length; //x contains 4.
coins.length = 3 // "Quarter" was removed from array.

//Display the contents of the array.
document.write("The coins array contains: ", coins.join(','));

// -->
</script>
```

Array.pop()

JavaScript 1.2+, ECMAScript 3E+, JScript 5.5+
 Nav4+, NES3+, IE5.5+, Opera5+

Syntax

```
array.pop()
```

Description

The `pop()` method “pops” elements off the end of the array by deleting the last element of the array and setting the array’s `length` property to one less than its current value. The element popped off the end of the array is returned from the method. Table 7.12 shows the return value associated with this method.

Table 7.12 Arguments and Return Values Associated with `pop()`

Type	Item	Description
Returns		The last element in the array is returned from the method.

Example

In Listing 7.69, an array of pages is created to represent a stack of papers. The `pop()` method removes and returns the top-most paper. After the `pop()` method is executed, the variable `currentPaper` contains “Page3”, and the array’s `length` property is 2.

Listing 7.69 Using the `pop()` Method to Remove Elements from the End of an Array

```
<script type="text/javascript" language="JavaScript">
<!--
```

Listing 7.69 Continued

```
//Create an array of papers and remove the top page.
pileOfPapers = new Array("Page1","Page2","Page3");
currentPaper = pileOfPapers.pop(); //Removed Page3
document.write(currentPaper," was removed from the pile.");

// -->
</script>
```

Array.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera5+

Syntax

`Array.prototype.property`

`Array.prototype.method`

Description

The prototype property allows you to add new properties and methods (designated as property/method in the Syntax section) to the Array object that can be used throughout your code.

Example

In Listing 7.70, the prototype property is used to provide a `pop()` method for working with arrays. Even though a `pop()` method is already available, a new `pop()` method will be created. This method overrides the functionality of the `Array.pop()` method. The new `pop()` method is used at the end of the code to remove an the "Peach" element from the array of juice flavors. The contents of the *flavorArray* are displayed before and after the new `pop()` method is executed to show that the "Peach" flavor was removed.

Listing 7.70 Assigning a New Method to the Array Object with the Prototype Property

```
<script type="text/javascript" language="JavaScript">
<!--

//This function removes the last element in the array. This last
//element is returned from the function.
function pop()
{
    if(this.length != 0)
    {
        var lastElement = this[this.length-1]; //Get last element.
        this.length = this.length-1;           //Remove last element from array.
        return(lastElement);                   //Return the last element.
```



```

    }
}

//Make the pop() function available to all Array objects.
//This will override the pop() method provided by the Array object in Netscape.
Array.prototype.pop = pop;

//Create an Array of juice flavors.
var flavorArray = new Array("Strawberry","Blueberry","Peach");

//Display the contents of the flavor array.
document.write("The flavor array initially contains: ");
document.write(flavorArray.join(', '), "<br>");

//Remove Peach from the array.
var removedElement = flavorArray.pop();
document.write(removedElement, " was removed from the flavor array.<br>");

//Display the contents of the flavor array after the pop() method.
document.write("The flavor array now contains: ");
document.write(flavorArray.join(', '), "<br>");

// -->
</script>

```

Array.push()

JavaScript 1.2+, ECMAScript 3E+, JScript 5.5+
Nav4+, NES3+, IE5.5+, Opera5+

Syntax

`array.push (arg1,...argN)`

Description

The `push()` method “pushes” the elements specified in the parameter list on to the end of the array in the order they were listed. Table 7.13 shows the arguments and return values associated with this method.

Table 7.13 Arguments and Return Values Associated with `push()`

Type	Item	Description
Arguments	<code>arg1,...argN</code>	One or more elements to be added to the end of the array
Returns		The last element added to the end of the array, which is also the last argument in the parameter list

Example

In Listing 7.71, an array of pages is created to represent a stack of papers. The `push()` method puts two more pages on the end of the array. After the `push()` method is executed, the variable `currentPaper` contains "Page4", and the array's `length` property is 4.

Listing 7.71 *Using the `push()` Method to Add Elements to the End of an Array*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array of papers.
pileOfPapers = new Array("Page1","Page2");

//Add 2 more pages to the end of the array.
currentPaper = pileOfPapers.push("Page3","Page4");

//Display the papers in the pile.
document.write(pileOfPapers.join(',')+" are in the pile.");

// -->
</script>
```

Array.reverse()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera5+

Syntax

```
array.reverse()
```

Description

The `reverse()` method reverses the order of the elements in the array according to the array index numbers.

Example

Listing 7.72 creates an array representing a line of people. The `reverse()` method is called to reverse the order of the names so that Polly is the first element, Leslie is the second element, and Cheryl is the last element.

Listing 7.72 *Reversing Element Positions in an Array with the `reverse()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create an array of names representing people in a grocery store line.
lineOfPeople = new Array("Cheryl","Leslie","Polly");
lineOfPeople.reverse(); //Reverse the items in the array.

//Display the names in the array. Notice the reversed ordering.
document.write("lineOfPeople[0]=" ,lineOfPeople[0], "<br>");
document.write("lineOfPeople[1]=" ,lineOfPeople[1], "<br>");
document.write("lineOfPeople[2]=" ,lineOfPeople[2], "<br>");

// -->
</script>
```

Array.shift()

JavaScript 1.2+, ECMAScript 3E+, JScript 5.5+
Nav4+, NES3+, IE5.5+, Opera5+

Syntax

```
array.shift()
```

Description

The `shift()` method deletes and returns the first element of the array. Once deleted, all the remaining elements are shifted down one spot, so the first position is filled by the element that was previously in the second position. Table 7.14 shows the return value associated with this method.

Table 7.14 Return Value Associated with `shift()`

Type	Item	Description
Returns		The first element of the array, before the elements are shifted, is returned from the method.

Example

Listing 7.73 creates an array representing people waiting for a table at a restaurant. The `shift()` method pulls the first name off the list and shifts all the other names down one position. After the `shift()` method is executed, the variable `personToSeat` now contains Kent. Jon is shifted to `lineOfPeople[0]` and Jeremy to `lineOfPeople[1]`.

Listing 7.73 Removing the First Element from an Array with the `shift()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array representing a line of people at a restaurant.
lineOfPeople = new Array("Kent","Jon","Jeremy");
personToSeat = lineOfPeople.shift(); //Kent pulled from array
```

Listing 7.73 Continued

```
//Display name of person removed from array.
document.write("Please seat ",personToSeat,"<br>");

//Display people left in the array.
document.write("People waiting for a seat: ",lineOfPeople.join(', '));
// -->
</script>
```

Array.slice()

JavaScript 1.2+, ECMAScript 3E+, JScript 3.0+
Nav4+, NES3+, IE 4+, Opera5+

Syntax

```
array.slice (start)
array.slice(start, stop)
```

Description

The slice() method returns a new array that contains the elements of the original array starting at position start and ending at the element position *before* stop. If no stop position is specified, the new array will contain the elements of the original array, starting at the position stated in start through the end of the array. Table 7.15 lists the arguments and return values associated with this method.

Table 7.15 Arguments and Return Values Associated with slice()

Type	Item	Description
Arguments	start	The position in the array where the slice is to begin. Negative numbers can be used to count from the last element to the first. For example, -1 is the last element in the array, and -2 is the second to the last element in the array.
	stop	The position in the array where the slice is to stop. Similar to the start parameter, the stop parameter can be negative.
Returns		A new array is returned from this method that contains the elements of the original array from index positions specified by start and stop.

Example

In Listing 7.74, an array of numbers is created. A new array of numbers is derived from the original array of numbers using the slice() method. After the slice() method is executed, the array newNumArray contains the elements 23 and 759.

Listing 7.74 *Selecting a Subsection from an Array with the slice() Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array of 4 numbers and display the contents.
numArray = new Array(345,23,759,5);
document.write("numArray contains the numbers: ",numArray.join(', '),"<br>");

//Create a new array from part of the original array.
newNumArray = numArray.slice(1,3);    // new array contains [23,759].

//Display the contents of the new array.
document.write("newNumArray contains the numbers: ",newNumArray.join(', 
'),"<br>");

// -->
</script>
```

Array.sort()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
 Nav3+, NES3+, IE 4+, Opera5+

Syntax

```
array.sort()
array.sort (function)
```

Description

The `sort()` method rearranges the elements of the array based on a sorting order. Table 7.16 lists the argument associated with this method. If the method has no parameters, JavaScript attempts to convert all the elements of the array to strings and then sort them alphabetically. If the array should be sorted some other way, a function must be provided to handle the new sorting algorithm.

Table 7.16 *Argument Associated with sort()*

Type	Item	Description
Argument	function	A function designated to handle the sorting of the array.

As mentioned before, if the array should be sorted some other way than javascriptically, a function must be provided to handle the new sorting algorithm. The function specified must operate based on the following rules:

- The function must accept two arguments that are to be compared.
- The function must return a number indicating the order of the two arguments in relation to each other.

- If the first argument should appear before the second argument, a number less than zero should be returned from the function.
- If the first argument should appear after the second argument, a number greater than zero should be returned from the function.
- If both arguments are equivalent, zero should be returned from the function.

When the function specified by the `sort()` method returns zero, signifying that the arguments are equal, the arguments remain in the same order relative to each other after the function has been called.

Example

To help solidify how this method operates, Listing 7.75 demonstrates how to use the method to sort an array based on the number of characters in each argument. Notice how the `sort()` method changes the order of the elements of the array in Figure 7.5.

Listing 7.75 Sorting an Array Based on Argument Lengths

```
<script type="text/javascript" language="JavaScript">

<!--
//Display the contents of an array.
function contentsOfArray(theArray)
{
    document.write("the array contains:<br>");
    //Access each element in the array.
    for(i=0; i<theArray.length; i++)
    {
        document.write("Position ",i," = ",theArray[i],"<br>");
    }
}

//Sort arguments based on their length.
function sortOnArgLen(arg1,arg2)
{
    if(arg1.length < arg2.length)
        return -1;
    if(arg1.length > arg2.length)
        return 1;
    if(arg1.length == arg2.length)
        return 0;
}

//Create and display an array of shapes.
shapes = new Array("triangle","rectangle","square");
document.write("Before the sort method ");
contentsOfArray(shapes);
```

```
//Sort the array.
shapes.sort(sortOnArgLen);
document.write("<br>After the sort method ");
contentsOfArray(shapes);
// -->
</script>
```

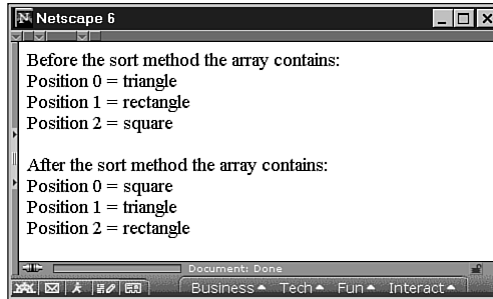


Figure 7.5

Sort the array elements based on the character length of each element.

Array.splice()

JavaScript 1.2+, ECMAScript 3E+, JScript 5.5+
Nav4+, IE5.5+, Opera5+

Syntax

`array.splice(start, delete, arg3, ..., argN)`

Description

The `splice()` method provides a way for elements to be either added or deleted from the array. When the `delete` parameter contains a number other than zero, the elements beginning at `start` and ending at `start+delete` are deleted from the array. If `delete` is zero, no elements are deleted. All elements from `start` to the end of the array are deleted when `delete` is not specified. If arguments follow the `delete` parameter, they are added to the array as elements beginning at the position specified by `start`. Existing elements are shifted up to allow room for the new elements.

NOTE

There was a bug in Navigator 4 when just one element was deleted from an array. Rather than returning an array that contained the deleted element, the actual element was returned from the method. In addition, when no elements were deleted from the array, `null` was returned instead of an empty array.

Table 7.17 lists the arguments and return values associated with the `splice()` method.

Table 7.17 *Arguments and Return Values Associated with splice()*

Type	Item	Description
Arguments	start	The position in the array where the slice is to begin.
	delete	The number of elements to be deleted from the array, beginning at the position specified by start.
	arg3,...,argN	New array elements to be inserted into the array, starting at the position specified by start.
Returns		If any elements are deleted from the array, they are returned from the method as an array.

Example

To help understand how the splice() method works, Listing 7.76 uses the delete and insert capabilities of the method to simulate a food order at a restaurant. In Figure 7.6, notice how the hamburger was replaced with a hot dog based on the customer’s change in appetite.

Listing 7.76 *Using splice() to Simulate a Food Order*

```
<script type="text/javascript" language="JavaScript">
<!--
//Display the current order.
function printOrder(theArray)
{
    document.write("The current order is:<br>");
    //Access each element in the array.
    for(i=0; i<theArray.length; i++)
    {
        document.write("- ",theArray[i],"<br>");
    }
}

//Create and display a food order.
foodOrder = new Array("hamburger","fries","drink");
document.write("<h3>The initial order taken</h3>");
printOrder(foodOrder);

//Replace the hamburger with a hotdog.
foodOrder.splice(0,1,"hotdog");
document.write("<h3>The customer wants a hotdog ");
document.write("instead of the hamburger.</h3>");

//Print the new order.
printOrder(foodOrder);
// -->
</script>
```

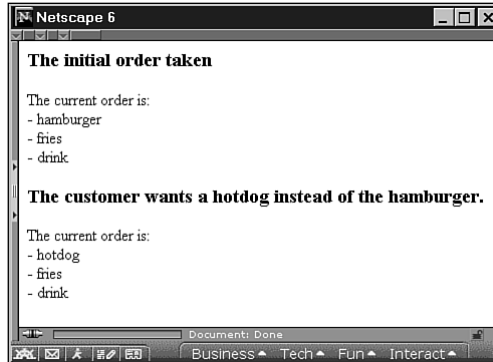



Figure 7.6

The food order changed using the `splice()` method.

Array.toSource()

JavaScript1.3+, JScript3.0

Nav4.06+, IE4

Syntax

`array.toSource()`

Description

The `toSource()` method returns one string representing the source of the Array object. The string that is returned contains all the elements in the array separated with commas. The entire string is enclosed with brackets (`[]`) to show it is an array. If another array is contained within an array, its contents are also part of the string with its own set of brackets.

Example

Listing 7.77 creates an Array object of colors and an Array object of numbers. The array of numbers is included in the array of colors. The `toSource()` method is then applied to the Array object to return the source string.

```
[ "Blue", "Green", "Red", [3, 6, 7] ]
```

Listing 7.77 Accessing the source of an Array object with the `toSource()` Method

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create an array of numbers.
```

```
numbers = new Array(3,6,7);
```

Listing 7.77 Continued

```
//Create an array of colors and a subarray of numbers.
colors = new Array("Blue","Green","Red",numbers);

//Force JavaScript to display the array contents in a string.
aString = colors.toSource(); //Create a string from the array contents.
document.write(aString); //Display the string.

// -->
</script>
```

Array.toString()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, IE 4+, Opera5+

Syntax

```
array.toString()
```

Description

The `toString()` method returns one string that contains all the elements in the array separated with commas. You might be wondering why anyone would need this method when the `join()` method can do this and more. The reason is that the `toString()` method is what JavaScript uses to automatically convert an array to a string when the array is used in string context.

CAUTION

JavaScript 1.2 did not use commas to separate the elements. Instead, each element was enclosed in square brackets.

Table 7.18 shows the return value associated with the `toString()` method.

Table 7.18 Return Value Associated with `toString()`

Type	Item	Description
Returns		A string that contains all the elements of the array

Example

Listing 7.78 creates an array of colors and then uses the `toString()` method to put the contents of the array into a string which is then displayed in the browser.

Listing 7.78 Forcing JavaScript to Use an Array's `toString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create an array of colors.
colors = new Array("Blue","Green","Red");

//Force JavaScript to display the array contents in a string.
document.write(colors.toString()); //returns "Blue,Green,Red"
// -->
</script>
```

Array.unshift()

JavaScript 1.2+, ECMAScript 3E+, JScript 5.5+
 Nav4+, NES3+, Opera5+

Syntax

```
array.unshift(arg1,...argN)
```

Description

The `unshift()` method adds the arguments listed in the parameter list to the front of the array as new elements. Existing elements are shifted up to allow room for the new elements. Table 7.19 lists the arguments and return values associated with this method.

Table 7.19 Arguments and Return Values Associated with `unshift()`

Type	Item	Description
Arguments	<code>arg1,...argN</code>	Elements to be added to the array
Returns		The length of the array after adding the new elements.

Example

Listing 7.79 creates an array of school grades. Two new grades, 100 and 93, are added to the front of the array using the `unshift()` method. After the `unshift()` method has executed, the `grades` array contains [100,93,95,87], and `newLength` contains the new length of the array, 4.

Listing 7.79 Adding Elements to the Front of an Array Using the `unshift()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array of test grades.
grades = new Array(95,87);

//Add two more grades to the array.
newLength = grades.unshift(100,93);

//Display the grades stored in the array.
for(i=0; i<newLength; i++)
```

Listing 7.79 Continued

```
{
    document.write("grades[" + i + "]=",grades[i],"<br>");
}

// -->
</script>
```

Array.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

Array.unwatch (property)

Description

The `unwatch()` method of the `Array` object, is used to turn off the watch on a particular property specified by *property*.

Example

Listing 7.80 shows how the `unwatch()` method is used to stop watching the user defined property *p*.

Listing 7.80 Example of the `unwatch()` method of the `Array` object

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myArray." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myArray = new Array();
Array.prototype.p = 1;
myArray.watch("p",alertme);

myArray.p = 2;
myArray.unwatch("p");
myArray.p = 3;
myArray.watch("p",alertme);
myArray.p = 4;
// -->
</script>
```

Array.valueOf()

JavaScript1.1+, ECMAScript 1E+, JScript3.0+
Nav3+, NES2+, IE4+, Opera3+

Syntax

`array.valueOf()`

Description

The `valueOf()` method returns the primitive value of the object. In terms of an instance of an Array object, this method returns the array elements separated by commas. If an array contains another array, the contents are flattened when this method is used.

Example

Listing 7.81 creates an Array object of colors and an Array object of numbers. The array of numbers is included in the array of colors. Because the `valueOf()` method returns the actual elements in the array, "Blue,Green,Red,3,6,7" is written to the browser.

Listing 7.81 Using the `valueOf()` Method to Return the Value of the Array *Object*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create an array of numbers.
numbers = new Array(3,6,7);

//Create an array of colors and a subarray of numbers.
colors = new Array("Blue","Green","Red",numbers);

//Display the primitive value of the array.
document.write(colors.valueOf()); //Display the array elements.

// -->
</script>
```

Array.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`Array.watch(property,function)`

Description

The `watch()` method of the `Array` object is used to turn on the watch of a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 7.82 shows how the `watch()` method is used to start watching the user-defined property `p`.

Listing 7.82 *Example of the* `watch()` *Method of the* `Array` *Object*

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myArray." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myArray = new Array();
Array.prototype.p = 1;
myArray.watch("p",alertme);

myArray.p = 2;
// -->
</script>
```

Automation

JScript3.0+
IE4+

Syntax

Core JScript Object

Description

Automation objects are objects that are connected to other applications or programming tools through automation interfaces. These objects give JScript developers access to properties and methods applications from within the JScript code. See `ActiveXObject()` and `GetObject()` for details on using automation objects.

Example

See `ActiveXObject()` and `GetObject()` for examples of using automation objects.

boolean

JavaScript1.2, ECMAScript 2E+, JScript 5+
Nav4+, NES3+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The `boolean` keyword has not been implemented in server-side JavaScript to date, but has been reserved for future use. Note that this keyword is not the same as the `Boolean` object.

Example

This keyword has not been implemented; therefore, no example is provided.

Boolean

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera5+

Syntax

```
var variable = new Boolean(value)
var variable = Boolean(value)
```

Description

The `Boolean` object is a wrapper object that holds a primitive boolean value, as well as provides a method for converting the value to a string. A primitive boolean can have only one of two states: `true` or `false`. Internally, JavaScript uses the number `1` to represent `true` and `0` to represent `false`, but provides the `toString()` method to return the strings `"true"` and `"false"`.

A `Boolean` object is created with the `Boolean()` constructor and the `new` operator or by the `Boolean()` function. The argument, return value, and method associated with this object are listed in Table 7.20.

Table 7.20 The Argument, Return Value, Properties, and Methods Associated with the `Boolean` Object

Type	Item	Description
Argument	value	The value to be converted to a Boolean value and stored in the object. The values <code>null</code> , <code>NaN</code> , <code>"</code> (empty string), and <code>0</code> (zero) are converted to <code>false</code> . All other values (including the string <code>"false"</code>) are converted to <code>true</code> .
Returns		If the <code>new</code> operator is used, the new <code>Boolean</code> object is returned. If the <code>Boolean()</code> function is used, the primitive Boolean value is returned.
Properties	constructor	Specifies the function that creates the <code>Boolean</code> object's prototype.
	prototype	Represents the prototype of this class.

Table 7.20 Continued

Type	Item	Description
Methods	toSource	Returns a string representation of the Boolean object.
	toString()	This method returns a string representation of the primitive Boolean value stored in the object. If the object contains true, the string "true" is returned. Similarly, if the object contains false, the string "false" is returned.
	unwatch()	Removes a watch point.
	valueOf()	Returns a Boolean value contained in the object.
	watch	Sets a watch point.

Example

In Listing 7.83, a Boolean object and a primitive Boolean value are created. There are a couple of key points to notice when examining the results generated by the code, shown in Figure 7.7. First, the Boolean() constructor converts the string "false" to the Boolean value true. Second, boolObj is a Boolean object, whereas boolVal is just a variable holding a primitive Boolean value.

Listing 7.83 A Boolean Object Versus a Primitive Boolean Value

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Boolean object.
boolObj = new Boolean("false");
document.write("boolObj = ",boolObj);           //Display true.
document.write(" [",typeof boolObj,"]<br>");      //Display object.

//Create a primitive boolean value.
boolVal = Boolean(false);
document.write("boolVal = ",boolVal);           //Display false.
document.write(" [",typeof boolVal,"]");         //Display boolean.

// -->
</script>
```

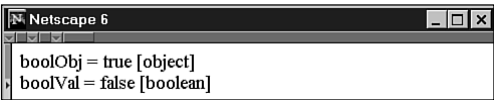


Figure 7.7

Boolean object versus primitive boolean value.

Boolean.constructor

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

Boolean.constructor

Description

The constructor property of the Boolean object is used to create the object's prototype.

Example

Listing 7.84 demonstrates the constructor property as it relates to the Boolean object.

Listing 7.84 *The Boolean's* constructor **Property**

```
<script type="text/javascript" language="JavaScript">
<!--
// Create a new boolean object using the constructor property.
myBoolean = new Boolean(true);
if(myBoolean.constructor == Boolean){
    document.write("Object was created");
}
// -->
</script>
```

Boolean.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, IE 4+, Opera5+

Syntax

Boolean.prototype.property

Boolean.prototype.method

Description

The prototype property allows you to add new properties and methods to the Boolean object that can be used throughout your code.

Example

In Listing 7.85, the prototype property is used to create a new method, called `letter()`, which can be used by all Boolean objects. The `letter()` method uses the `letterBoolean()` function to return true or false based on the status of the object.

Listing 7.85 Assigning a New Method to the Boolean Object with the prototype *Property*

```
<script type="text/javascript" language="JavaScript">
<!--

//This function returns the string "T" or "F" based on the value stored in the
//Boolean object that uses this function.
function letterBoolean()
{
    if(this == true)
        return("T");
    else
        return("F");
}

//Make the letterBoolean function available to all Boolean objects.
Boolean.prototype.letter = letterBoolean;

//Create a Boolean object with an initial setting of true.
var myBooleanObj = new Boolean(true); //myBooleanObj equal to true

//Display the state of the Boolean object using the letter method.
document.write("myBooleanObj is set to ",myBooleanObj.letter()); //Return "T"

// -->
</script>
```

Boolean.toSource()

JavaScript1.3+, JScript3.0
Nav4.06+, IE4

Syntax

```
boolean.toSource()
```

Description

The `toSource()` method returns one string representing the source of the Boolean object. The string that is returned is enclosed in parentheses.

Example

Listing 7.86 creates a Boolean object to represent true. The `toSource()` method is then applied to the Boolean object to return the source string "(new Boolean(true))".

Listing 7.86 Accessing the Source of a Boolean Object with the *toSource()* Method

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create a Boolean object representing true.
bool = new Boolean(1);

//Display the source of the Boolean object.
document.write(bool.toSource());

// -->
</script>
```

Boolean.toString()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, IE 4+, Opera5+

Syntax

boolean.toString()

Description

The toString() method returns the string representation ("true" or "false") of the primitive Boolean value stored in the Boolean object. The return value associated with this object is listed in Table 7.21.

Table 7.21 *Return Value Associated with* toString()

Type	Description
Returns	If true, the string "true" is returned. If false, the string "false" is returned.

Example

In Listing 7.87, the toString() method is used to force a comparison of strings rather than Boolean values. Without the toString() method, the if comparison would find the Boolean value not equal to the string value.

Listing 7.87 *Force the Desired Comparison Using* Boolean's toString() *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Boolean object that contains the boolean value "false".
boolObj = new Boolean(false);

//Force JavaScript to convert the boolObj object to the string "false"
//before comparing to the string "false".
if(boolObj.toString() == "false")    //Evaluates to true,
    alert("EQUAL");                 //so display "EQUAL" on screen.
```

Listing 7.87 Continued

```
else
    alert("NOT Equal");

// -->
</script>
```

Boolean.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

Boolean.unwatch (property)

Description

The `unwatch()` method of the `Boolean` object is used to turn off the watch on a particular property specified by *property*.

Example

Listing 7.88 shows how the `unwatch()` method is used to stop watching the user-defined property `p`.

Listing 7.88 Example of the `unwatch()` method of the `Boolean` Object

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myBoolean." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myBoolean = new Boolean();
Boolean.prototype.p = 1;
myBoolean.watch("p",alertme);

myBoolean.p = 2;
myBoolean.unwatch("p");
myBoolean.p = 3;
myBoolean.watch("p",alertme);
myBoolean.p = 4;
// -->
</script>
```

Boolean.valueOf()

JavaScript1.1+, ECMAScript 1E+, JScript3.0+, Nav3+, NES2+, IE4+, Opera3+

Syntax

`boolean.valueOf()`

Description

The `valueOf()` method returns the primitive value of the object. In terms of an instance of a `Boolean` object, this method returns a `Boolean` value contained in the object.

Example

Listing 7.89 creates a `Boolean` object representing `true`. Because the `valueOf()` method returns the `Boolean` value in the object, "true" is written to the browser.

Listing 7.89 Using the `valueOf()` Method to Return the Value of the `Boolean` Object

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create a Boolean object representing true.
bool = new Boolean(1);
```

```
//Display the source of the Boolean object.
document.write(bool.valueOf());
```

```
// -->
</script>
```

Boolean.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`Boolean.watch (property,function)`

Description

The `watch()` method of the `Boolean` object is used to turn on the watch of a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 7.90 shows how the `watch()` method is used to start watching the user-defined property `p`.

Listing 7.90 *Example of the watch() Method of the Boolean Object*

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myBoolean." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myBoolean = new Boolean();
Boolean.prototype.p = 1;
myBoolean.watch("p",alertme);

myBoolean.p = 2;
// -->
</script>
```

break

JavaScript 1.1+, ECMAScript 1E+, JScript 1.0+
Nav3+, IE 3+, Opera3+

Syntax

```
break label;
```

Description

The keyword `break` provides a way to exit out of loop structures and switch conditionals prematurely. Most of the time, the word `break` appears on a line by itself, but there are times when a label will follow the keyword (see Table 7.22). When a label is used, code execution completely breaks out of the area designated by `label` and proceeds to the code that follows the area. To label a statement, simply place the label name followed by a colon (`:`) in front of the code that needs to be broken out of during code execution.

Table 7.22 *Argument Associated with the break Keyword*

Type	Item	Description
Argument	label	A label that designates code from which to break.

Example

Listing 7.91 demonstrates the effect of using labels and `break` statements when working with nested loops. Figure 7.8 shows the result of executing this code.

Listing 7.91 *Using breaks and Labels*

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Loop through the outer loop (forLoop1) 4 times.
forLoop1:
for (var counter1 = 1; counter1 <= 5; counter1++)
{
    //Go through this inner loop 4 times for each time through the outer loop.
    for (var counter2 = 1; counter2 <= 5; counter2++)
    {
        //Display the values in each for loop's counter.
        document.write("Counter1=",counter1);
        document.write(" Counter2=",counter2,"<br>");

        //Break out of inner loop the 3rd time through the inner loop.
        if (counter2 == 3)
            break;

        //Break out of the outer loop the 3rd time through the outer loop.
        if (counter1 == 3)
            break forLoop1;
    }
}

document.write("All done!");

//-->
</script>
```

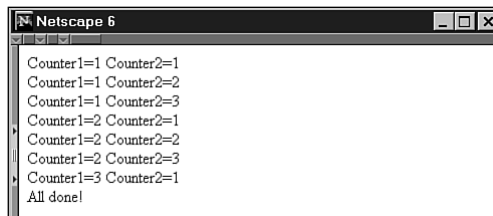


Figure 7.8

Using labels and the break statement in nested loops.

Notice how the break statement with no label, in Figure 7.8, breaks out of just the inner loop. When the break statement is used with a label, JavaScript knows at what level to break.

byte

JavaScript 1.2+, ECMAScript 2E+, JScript 5+
Nav4+, NES3+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The `byte` keyword has not been implemented in server-side JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

case

JavaScript1.2+, ECMAScript 2E+, JScript3.0+
Nav4+, NES3+, IE4+, Opera5+

Syntax

Reserved Keyword

Description

The `case` keyword has not been implemented in server-side JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

char

JavaScript1.2+, ECMAScript 2E+, JScript 5+
Nav4+, NES2+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The `char` keyword has not been implemented in server-side JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

class

JavaScript1.2+, ECMAScript 1E+, JScript3.0+
Nav4+, NES3+, IE4+, Opera5+

Syntax

Reserved Keyword

Description

The word `class` is reserved for future use, so there is no definition at this time.

Example

No example can be provided because `class` is reserved for future use.

const

JavaScript1.2, ECMAScript 1E+, JScript 5+
Nav4+, NES3+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The word `const` is reserved for future use, so there is no definition at this time.

Example

No example can be provided because `const` is reserved for future use.

continue

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, IE 4+, Opera3+

Syntax

```
continue label;
```

Description

The `continue` statement forces the execution of the code within a loop to continue at the beginning of the loop. Normally, the `continue` keyword appears on a line by itself, but there are times when a label will follow the keyword (see Table 7.23). When a label is used, code execution immediately jumps to the beginning of the loop designated by the label and begins executing code.

Table 7.23 *Argument Associated with the `continue` Keyword*

Type	Item	Description
Argument	label	A label that designates code to execute.

The beginning of a loop varies depending on the type of loop structure. Table 7.24 shows where each looping structure jumps to when a `continue` structure is encountered.

Table 7.24 *Where the `continue` Statement Jumps*

Looping Structure	Continue Jumps to
<code>for</code>	Expression in parentheses following <code>for</code> keyword
<code>while</code>	Expression in parentheses following <code>while</code> keyword
<code>do...while</code>	Expression in parentheses following <code>while</code> keyword
<code>for...in</code>	Next property name in object

 CAUTION

There is a bug in Navigator 4 that causes the expression in parentheses following the `while` keyword in a `do...while` loop to not get executed when jumped to using a `continue` statement. Instead, execution of code starts at the top of loop, after the `continue` statement. This problem can be avoided by using a `while` loop.

To label a statement, simply place the label name followed by a colon (`:`) in front of the code where code execution needs to continue.

Example

Listing 7.92 demonstrates the use of labels and `continue`. This example is a bit complicated, so take time to compare Listing 7.79 to the output in Figure 7.9. Notice how the phrase "Bottom of innerLoop" was not printed after the "Continue at top of innerLoop." because code execution jumped back to beginning of the inner-most loop. When a label was attached to the `continue` keyword, code execution jumped back to the beginning of the loop labeled `outerLoop`.

Listing 7.92 Using the `continue` Statement

```
<script type="text/javascript" language="JavaScript">
<!--

//Loop through the outerLoop twice.
outerLoop:
  for (var counter1 = 1; counter1 <= 2; counter1++)
  {
    document.write("Top of outerLoop.<br>");

    //Loop through the innerLoop twice.
    innerLoop:
      for (var counter2 = 1; counter2 <= 2; counter2++)
      {
        //Display the values in each for loop's counter.
        document.write("Top of innerLoop.<br>");
        document.write("Counter1=",counter1,"<br>");
        document.write("Counter2=",counter2,"<br>");

        //If this is the second time through the innerLoop,
        //don't go any further and jump back to the top of the innerLoop.
        if (counter2 == 2)
        {
          document.write("Continue at top of innerLoop.<br>");
          continue;
        }

        //If this is the second time through the outerLoop,
        //don't go any further and jump back to the top of the outerLoop.
```

```

    if (counter1 == 2)
    {
        document.write("Continue at top of outerLoop.<br>");
        continue outerLoop;
    }
    document.write("Bottom of innerLoop.<br>");
}
document.write("Bottom of outerLoop.<br>");
}

document.write("All done!");

//-->
</script>

```

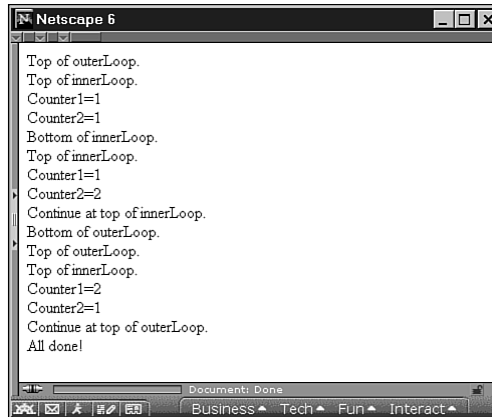


Figure 7.9

The result of using the continue statement.

Date

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```

var variable = new Date();
var variable = new Date(milliseconds1);
var variable = new Date(string)
var variable = new Date(year, month, day, hours, minutes, seconds,
milliseconds2)

```

Description

The Date object represents all aspects of a date and time from year to milliseconds. If arguments are provided when creating the Date object, the new object will contain the

date specified; otherwise the object will be set to the current date. The arguments and methods are listed in Table 7.25.

CAUTION

If the `Date()` constructor can also be called as a function by excluding the new operator. With this is done a string representation of the date is returned rather than a `Date` object

Table 7.25 Arguments and Methods Associated with the `Date` Object

Type	Item	Description
Argument	constructor	Specifies the function that creates the <code>Boolean</code> object's prototype.
	milliseconds1	The desired date is calculated from the number of milliseconds between midnight January 1, 1970 GMT and the desired date.
	string	The desired date is calculated from the string representation. The format of the string should match the <code>parse()</code> method of the <code>Date</code> object.
	year	A four-digit representation of the year.
	month	The month represented as an integer, where 0 represents January and 11 represents December.
	day	The day of the month represented as an integer from 1 to 31. Optional argument.
	hours	The hour represented as an integer where 0 represents 12 a.m. (midnight) and 23 represents 11 p.m. Optional argument.
	minutes	The minutes in the hour represented as an integer from 0 to 59. Optional argument.
	seconds	The seconds in the minute represented as an integer from 0 to 59. Optional argument.
	milliseconds2	The milliseconds in the second represented as an integer from 0 to 999. Optional argument.
Methods	<code>getDate()</code>	Returns the day of the month.
	<code>getDay()</code>	Returns the day of the week.
	<code>getFullYear()</code>	Returns the year in local time with four digits.
	<code>getHours()</code>	Returns the hour.
	<code>getMilliseconds()</code>	Returns the milliseconds.

<code>getMinutes()</code>	Returns the minutes.
<code>getMonth()</code>	Returns the month.
<code>getSeconds()</code>	Returns the seconds.
<code>getTime()</code>	Returns the date and time in milliseconds.
<code>getTimezoneOffset()</code>	Returns the time zone offset from GMT in minutes.
<code>getUTCDate()</code>	Returns the day of the month converted to universal time.
<code>getUTCDay()</code>	Returns the day of the week converted to universal time.
<code>getUTCFullYear()</code>	Returns a four-digit representation of the year converted to universal time.
<code>getUTCHours()</code>	Returns the hour converted to universal time.
<code>getUTCMilliseconds()</code>	Returns the milliseconds converted to universal time.
<code>getUTCMinutes()</code>	Returns the minutes converted to universal time.
<code>getUTCMonth()</code>	Returns the month converted to universal time.
<code>getUTCSeconds()</code>	Returns the seconds converted to universal time.
<code>getVarDate()</code>	Returns the date in VT_DATE format.
<code>getYear()</code>	Returns the year as either four digits or two digits.
<code>parse()</code>	Converts a string, representing a date and time, into milliseconds.
<code>setDate()</code>	Sets the day of the month.
<code>setFullYear()</code>	Sets the year as a four-digit number.
<code>setHours()</code>	Sets the hour.
<code>setMilliseconds()</code>	Sets the milliseconds.
<code>setMinutes()</code>	Sets the minutes.
<code>setMonth()</code>	Sets the month.
<code>setSeconds()</code>	Sets the seconds.
<code>setTime()</code>	Sets the date and time from a millisecond representation of a date and time.
<code>setUTCdate()</code>	Sets the day of the month in universal time.
<code>setUTCFullYear()</code>	Sets the year as a four-digit number in universal time.
<code>setUTCHours()</code>	Sets the hour in universal time.
<code>setUTCMilliseconds()</code>	Sets the milliseconds in universal time.
<code>setUTCMinutes()</code>	Sets the minutes in universal time.
<code>setUTCMonth()</code>	Sets the month in universal time.

Table 7.25 Continued

Type	Item	Description
	setUTCSeconds()	Sets the seconds in universal time.
	setYear()	Sets the year as either a four-digit number or a two-digit number.
	toGMTString()	Returns the data and time as a string in universal time (GMT).
	toLocalString()	Returns the date and time as a string in local time format.
	toSource()	Returns the source of the Date object.
	toString()	Returns the date and time as a string in local time.
	toUTCString()	Returns the data and time as a string in universal time (GMT).
	unwatch	Removes a watchpoint.
	UTC()	Converts a universal date and time (GMT) to milliseconds.
	watch	Sets a watchpoint.

Example

Listing 7.93 demonstrates how to create Date objects using various arguments in the Date() constructor.

Listing 7.93 Creating Date Objects

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a new Date object using all arguments.
theDate1 = new Date(2002,9,29,20,5,8,10);

//Create a new Date object using just milliseconds.
theDate2 = new Date(theDate1.getTime());

//Create a new Date object using a string.
theDate3 = new Date(theDate1.toString());

//Display all the date objects.
document.write("theDate1-> ",theDate1.toString());
document.write("<br>theDate2-> ",theDate2.toString());
document.write("<br>theDate3-> ",theDate3.toString());

// -->
</script>
```

Date.constructor

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera3+

Syntax

`Date.constructor`

Description

The constructor property of the Date object is used to create the object's prototype.

Example

Listing 7.94 demonstrates the constructor property as it relates to the Date object.

Listing 7.94 The Date's constructor Property

```
<script type="text/javascript" language="JavaScript">
<!--
// Create a new date object using the constructor property.
myDate = new Date();
if(myDate.constructor == Date){
    document.write("Object was created");
}
// -->
</script>
```

Date.getDate()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`date.getDate ()`

Description

The `getDate()` method returns the day of the month expressed as an integer from 1 to 31.

Example

The code in Listing 7.95 displays the current day of the month expressed as an integer from 1 to 31 using the `getDate()` method.

Listing 7.95 Using the `getDate()` Method to Return the Day of the Month

```
<script type="text/javascript" language="JavaScript">
<!--
```

Listing 7.95 Continued

```
//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the date.
document.write("The date is ",theDate.getDate());

// -->
</script>
```

Date.getDay()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getDay()
```

Description

The `getDay()` method returns the day of the week expressed as an integer from 0 (Sunday) to 6 (Saturday).

Example

The code in Listing 7.96 uses the `getDay()` method to return the day of the week expressed as an integer. This number is then converted to a string representation of the day of the week.

Listing 7.96 Using the `getDay()` Method to Return the Day of the Week

```
<script type="text/javascript" language="JavaScript">
<!--

//This function converts the day from a number to
//a string and returns the string.
function getDayString(num)
{
    var day;    //Create a local variable to hold the string.
    switch(num)
    {
        case 0:
            day="Sunday";
            break;
        case 1:
            day="Monday";
            break;
        case 2:
            day="Tuesday";
            break;
```



```
    case 3:
        day="Wednesday";
        break;
    case 4:
        day="Thursday";
        break;
    case 5:
        day="Friday";
        break;
    case 6:
        day="Saturday";
        break;
    default:
        day="Invalid day";
}
return day;
}

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the day.
document.write("Today is ",getDayString(theDate.getDay()));
// -->
</script>
```

Date.getFullYear()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getFullYear()
```

Description

The `getFullYear()` method returns the year in local time as a full four-digit number.

Example

The code in Listing 7.97 displays the year using the `getFullYear()` method.

Listing 7.97 Using the `getFullYear()` Method to Return the Year

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//Create a Date object that contains the current date and time.
theDate = new Date();
```

Listing 7.97 Continued

```
//Display the full year.
document.write("The year is ",theDate.getFullYear());

// -->
</script>
```

Date.getHours()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getHours()
```

Description

The `getHours()` method returns the hour portion of the date expressed as an integer from 0 (12:00 a.m. midnight) to 23 (11:00 p.m.).

Example

The code in Listing 7.98 displays the current hour as an integer from 0 to 23 using the `getHours()` method.

Listing 7.98 Using the `getHours()` Method to Return the Current Hour

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the hour.
document.write("The hour is ",theDate.getHours());

// -->
</script>
```

Date.getMilliseconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getMilliseconds()
```

Description

The `getMilliseconds()` method returns the millisecond portion of the date expressed as an integer from 0 to 999.

Example

The code in Listing 7.99 displays the milliseconds in the Date object using the `getMilliseconds()` method.

Listing 7.99 Using the `getMilliseconds()` Method to Return the Current Millisecond

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the milliseconds.
document.write("The millisecond is ",theDate.getMilliseconds());

// -->
</script>
```

Date.getMinutes()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getMinutes()
```

Description

The `getMinutes()` method returns the minutes portion of the Date object expressed as an integer from 0 to 59.

Example

The code in Listing 7.100 displays the current minute using the `getMinutes()` method.

Listing 7.100 Using the `getMinutes()` Method to Return the Current Minute

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the minutes.
document.write("The minute is ",theDate.getMinutes());

// -->
</script>
```

Date.getMonth()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getMonth()
```

Description

The `getMonth()` method returns the month portion of the `Date` object expressed as an integer from 0 (January) to 11 (December).

Example

The code in Listing 7.101 displays the current month using the `getMonth()` method.

Listing 7.101 Using the `getMonth()` Method to Return the Current Month

```
<script type="text/javascript" language="JavaScript">
<!--

//This function converts the month from a number to
//a string and returns the string.
function getMonthString(num)
{
    var month;    //Create a local variable to hold the string.
    switch(num)
    {
        case 0:
            month="January";
            break;
        case 1:
            month="February";
            break;
        case 2:
            month="March";
            break;
        case 3:
            month="April";
            break;
        case 4:
            month="May";
            break;
        case 5:
            month="June";
            break;
        case 6:
            month="July";
```

```
        break;
    case 7:
        month="August";
        break;
    case 8:
        month="September";
        break;
    case 9:
        month="October";
        break;
    case 10:
        month="November";
        break;
    case 11:
        month="December";
        break;
    default:
        month="Invalid month";
    }
    return month;
}
```

```
//Create a Date object that contains the current date and time.
theDate = new Date();
```

```
//Display the month.
document.write("The month is ",getMonthString(theDate.getMonth()));
```

```
// -->
</script>
```

Date.getSeconds()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getSeconds()
```

Description

The `getSeconds()` method returns the seconds portion of the `Date` object expressed as an integer from 0 to 59.

Example

The code in Listing 7.102 displays the current seconds using the `getSeconds()` method.

Listing 7.102 Using the `getSeconds()` Method to Return the Current Second

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the seconds.
document.write("The second is ",theDate.getSeconds());

// -->
</script>
```

Date.getTime()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getTime()
```

Description

The `getTime()` method returns the equivalence of the `Date` object in milliseconds. The milliseconds are expressed as an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object.



TIP

It is much easier to compare two different dates as milliseconds by using the `getTime()` method, rather than having to examine the individual parts of the date.

Example

The code in Listing 7.103 displays, 9466848000000, the number of milliseconds that passed between midnight January 1, 1970 and midnight January 1, 2000 using the `getTime()` method.

Listing 7.103 Using the `getTime()` Method to Return the Milliseconds Between 1970 and 2000

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains midnight January 1, 1970.
Date1970 = new Date(1970,1,1,0,0,0,0);
```

```
//Create a Date object that contains midnight January 1, 2000.
date2000 = new Date(2000,1,1,0,0,0,0);
//Display the elapsed milliseconds.
document.write(date2000.getTime()-date1970.getTime());
document.write(" milliseconds passed between 1-1-1970 and 1-1-2000");

// -->
</script>
```

Date.getTimezoneOffset()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getTimezoneOffset()
```

Description

The `getTimezoneOffset()` method returns the difference between the time zones of local time and Greenwich Mean Time (GMT). This difference is returned as an integer representing the number of minutes between the time zone of the local time and GMT. Although this is a method of a `Date` object, the actual date and time associated with the date is irrelevant because the time zone difference is based on the environment settings in which the JavaScript code is run.

Example

The code in Listing 7.104 displays the time zone offset by using the `getTimezoneOffset()` method. Notice that when the date and time are changed, the time zone offset remains the same because the contents of the `Date` object are not used in the calculation.

Listing 7.104 Using the `getTimezoneOffset()` Method to Return the Time Zone Offset

```
<script type="text/javascript" language="JavaScript">
<!--

//Create 2 very different date objects.
aDate1 = new Date(1990,1,1,0,0,0,0);
aDate2 = new Date(1994,2,13,8,24,45,300);

//Display the timezone offsets.
document.write("The timezone offset of aDate1 is ");
document.write(aDate1.getTimezoneOffset()," minutes.<br>");
document.write("The timezone offset of aDate2 is "
document.write(aDate2.getTimezoneOffset()," minutes.");

// -->
</script>
```

Date.getUTCDate()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCDate()
```

Description

The `getUTCDate()` method returns the day of the month converted to universal time and expressed as an integer from 1 to 31.

Example

The code in Listing 7.105 displays the day of the month in universal time using the `getUTCDate()` method.

Listing 7.105 Using the `getUTCDate()` Method to Return the Day of the Month

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC date.
document.write("The UTC date is ",theDate.getUTCDate());

// -->
</script>
```

Date.getUTCDay()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera3+

Syntax

```
date.getUTCDay()
```

Description

The `getUTCDay()` method returns the day of the week converted to universal time and expressed as an integer from 0 (Sunday) to 6 (Saturday).

Example

The code in Listing 7.106 uses the `getUTCDay()` method to return the day of the week expressed as an integer in universal time. The number is then converted to a string equivalent to the day of the week.

Listing 7.106 *Using the `getUTCDay()` Method to Return the Day of the Week*

```

<script type="text/javascript" language="JavaScript">
<!--

//This function converts the day from a number to
//a string and returns the string.
function getDayString(num)
{
    var day;    //Create a local variable to hold the string.
    switch(num)
    {
        case 0:
            day="Sunday";
            break;
        case 1:
            day="Monday";
            break;
        case 2:
            day="Tuesday";
            break;
        case 3:
            day="Wednesday";
            break;
        case 4:
            day="Thursday";
            break;
        case 5:
            day="Friday";
            break;
        case 6:
            day="Saturday";
            break;
        default:
            day="Invalid day";
    }
    return day;
}

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC day.
document.write("The UTC day is ",getDayString(theDate.getUTCDay()));

// -->
</script>

```

Date.getUTCFullYear()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCFullYear()
```

Description

The `getUTCFullYear()` method returns the year as a full four-digit number converted to universal time.

Example

The code in Listing 7.107 displays the year in universal time using the `getUTCFullYear()` method.

Listing 7.107 Using the `getUTCFullYear()` Method to Return the Year

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the full UTC year.
document.write("The UTC year is ",theDate.getUTCFullYear());

// -->
</script>
```

Date.getUTCHours()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCHours()
```

Description

The `getUTCHours()` method returns the hour portion of the date, converted to universal time and expressed as an integer from 0 (12:00 a.m. midnight) to 23 (11:00 p.m.).

Example

The code in Listing 7.108 displays the current hour in universal time using the `getUTCHours()` method.

Listing 7.108 Using the `getUTCHours()` Method to Return the Current Hour

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the hour.
document.write("The UTC hour is ",theDate.getUTCHours());

// -->
</script>

```

Date.getUTCMilliseconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
 Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCMilliseconds()
```

Description

The `getUTCMilliseconds()` method returns the milliseconds portion of the date, converted to universal time and expressed as an integer from 0 to 999.

Example

The code in Listing 7.109 displays the current milliseconds in universal time using the `getUTCMilliseconds()` method.

Listing 7.109 Using the `getUTCMilliseconds()` Method to Return the Current Milliseconds

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC milliseconds.
document.write("The UTC millisecond is ",theDate.getUTCMilliseconds());

// -->
</script>

```

Date.getUTCMinutes()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
 Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCMinutes()
```

Description

The `getUTCMinutes()` method returns the minutes portion of the `Date` object, converted to universal time and expressed as an integer from 0 to 59.

Example

The code in Listing 7.110 displays the current minutes in universal time using the `getUTCMinutes()` method.

Listing 7.110 *Using the `getUTCMinutes()` Method to Return the Current Minutes*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC minutes.
document.write("The UTC minute is ",theDate.getUTCMinutes());

// -->
</script>
```

Date.getUTCMonth()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCMonth()
```

Description

The `getUTCMonth()` method returns the month portion of the `Date` object, converted to universal time and expressed as an integer from 0 (January) to 11 (December).

Example

The code in Listing 7.111 uses the `getUTCMonth()` method to return the current month, expressed as an integer, in universal time. The integer is converted to a string representation of the month.

Listing 7.111 *Using the `getUTCMonth()` Method to Return the Current Month*

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
//This function converts the month from a number to
//a string and returns the string.
function getMonthString(num)
{
    var month;    //Create a local variable to hold the string.
    switch(num)
    {
        case 0:
            month="January";
            break;
        case 1:
            month="February";
            break;
        case 2:
            month="March";
            break;
        case 3:
            month="April";
            break;
        case 4:
            month="May";
            break;
        case 5:
            month="June";
            break;
        case 6:
            month="July";
            break;
        case 7:
            month="August";
            break;
        case 8:
            month="September";
            break;
        case 9:
            month="October";
            break;
        case 10:
            month="November";
            break;
        case 11:
            month="December";
            break;
        default:
            month="Invalid month";
    }
    return month;
}
```

Listing 7.111 Continued

```
//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC month.
document.write("The UTC month is ",getMonthString(theDate.getUTCMonth()));

// -->
</script>
```

Date.getUTCSeconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.getUTCSeconds()
```

Description

The `getUTCSeconds()` method returns the seconds portion of the `Date` object, converted to universal time and expressed as an integer from 0 to 59.

Example

The code in Listing 7.112 displays the current second in universal time using the `getUTCSeconds()` method.

Listing 7.112 Using the `getUTCSeconds()` Method to Return the Current Second

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the UTC seconds.
document.write("The UTC seconds is ",theDate.getUTCSeconds());

// -->
</script>
```

Date.getVarDate()

JScript 3.0+
IE 4+

Syntax

```
date.getVarDate()
```

Description

The `getVarDate()` method returns the date value in `VT_DATE` format, which can be used with Visual Basic and VBScript when interacting with COM objects, ActiveX objects, and so on. The format is dependent on regional settings.

Example

The code in Listing 7.113 displays the date using the `getVarDate()` method.

Listing 7.113 Using the `getVarDate()` Method

```
<script type="text/jscript" language="JScript">
<!--
//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the date.
document.write("The date is ",theDate.getVarDate());
// -->
</script>
```

Date.getYear()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.getYear()
```

Description

The `getYear()` method returns the year portion of the `Date` object. Unfortunately, the year is represented as either a two-digit number or a four-digit number, depending on the browser version. For example, the year 1983 might be returned from the methods as 1983 or just 83.

NOTE

The `getYear()` method is obsolete. The `getFullYear()` method should be used instead of `getYear()`.

Example

The code in Listing 7.114 displays the year using the `getYear()` method associated with your specific browser.

Listing 7.114 Using the `getYear()` Method to Return the Current Year

```
<script type="text/javascript" language="JavaScript">
<!--
```

Listing 7.114 Continued

```
//Create a Date object that contains the current date and time.
theDate = new Date();

//Display the year.
document.write("The year is ",theDate.getYear());

// -->
</script>
```

Date.parse()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera5+

Syntax

Date.parse (date)

Description

The parse() method returns the time, represented in milliseconds between the date argument string and midnight, January 1, 1970, GMT. Notice that this method is associated with the "Date" object rather than a date that was declared. The string date should use the format written by the Date.toGMTString() method, which resembles the following:

Mon, 24 Oct 1982 12:03:27 -0200

The method will also accept strings that lack all or any portion of the time and time zone.

Example

The code in Listing 7.115 displays the milliseconds from 1970 to the string representation of the date and time.

Listing 7.115 Using the parse() Method

```
<script type="text/javascript" language="JavaScript">
<!--
//Convert the string representation of the date "Sun, 24 Oct 1982 12:03:27"
//to milliseconds and display the milliseconds.
document.write("The milliseconds of the string ";
document.write("'Sun, 24 Oct 1982 12:03:27': ");
document.write(Date.parse("Sun, 24 Oct 1982 12:03:27"));

// -->
</script>
```


Date.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES3+, IE 4+, Opera5+

Syntax

`Date.prototype.property`

`Date.prototype.method`

Description

The prototype property allows you to add new properties and methods to the Date object that can be used throughout your code.

Example

In Listing 7.116, the prototype property is used to create a new method, called `getDayString`, which can be used by all Date objects. The `getDayString()` method uses the `getDayString()` function to return a string representation of the day of the week (for example, Sunday, Monday, Tuesday,...). This new functionality is used to display the name of the current day.

Listing 7.116 Creating a New Method to Get the String Representation of the Day

```
<script type="text/javascript" language="JavaScript">
<!--

//This function returns a string representation of the day.
function getDayString()
{
    var day
    switch(this.getDay())
    {
        case 0:
            day="Sunday";
            break;
        case 1:
            day="Monday";
            break;
        case 2:
            day="Tuesday";
            break;
        case 3:
            day="Wednesday";
            break;
        case 4:
            day="Thursday";
            break;
```

Listing 7.116 Continued

```
    case 5:
        day="Friday";
        break;
    case 6:
        day="Saturday";
        break;
    default:
        day="Invalid day";
}
return(day);
}

//Make the getDayString function available to all Date objects.
Date.prototype.getDayString = getDayString;

//Create a Date object with current date and time.
var currentDate = new Date();

//Display the day as a string.
document.write("<h2>Today is ",currentDate.getDayString(),"</h2>");

// -->
</script>
```

Date.setDate()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`date.setDate(day)`

Description

The `setDate()` method sets the day of the month in the `Date` object to the argument `day`, an integer from 1 to 31. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the day of the month has been adjusted.

CAUTION

Prior to JavaScript 1.2, this method returned nothing.

Example

The code in Listing 7.117 displays the current date in milliseconds before setting the day to 4. Notice that the milliseconds specified after the `setDate()` method are the same as the result returned from the method.

Listing 7.117 Setting the Day of the Month with the setDate() Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the day of the month and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setDate returned ");
document.write(theDate.setDate(4)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>

```

Date.setFullYear()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
 Nav4+, IE 4+, Opera5+

Syntax

```
date.setFullYear(year)
```

Description

The setFullYear() method sets the year in the Date object to the argument year, a four-digit integer. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the Date object after the year has been adjusted.

Example

The code in Listing 7.118 displays the current date in milliseconds before setting the year to 2003. Notice that the milliseconds specified after the setFullYear() method are the same as the result returned from the method.

Listing 7.118 Setting the Day of the Week with the setFullYear() Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the year and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setFullYear returned ");

```

Listing 7.118 Continued

```
document.write(theDate.setFullYear(2003)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setHours()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

date.setHours (hours)

Description

The `setHours()` method sets the hour in the `Date` object to the argument `hours`, an integer from 0 (12:00 a.m. midnight) to 23 (11:00 p.m.). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the hour has been adjusted.

Example

The code in Listing 7.119 displays the current date in milliseconds before setting the hour to 7 (8:00 a.m.). Notice that the milliseconds specified after the `setHours()` method are the same as the result returned from the method.

Listing 7.119 Setting the Hour with the `setHours()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the hour and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setHours returned ");
document.write(theDate.setHours(7)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setMilliseconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

date.setMilliseconds(milliseconds)

Description

The `setMilliseconds()` method sets the milliseconds in the `Date` object to the argument `milliseconds`, an integer from 0 to 999. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the milliseconds have been adjusted.

Example

Listing 7.120 displays the current date in milliseconds before setting the milliseconds to 792. Notice that the milliseconds specified after the `setMilliseconds()` method are the same as the result returned from the method.

Listing 7.120 *Setting the Milliseconds with the* `setMilliseconds()` *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setMilliseconds returned ");
document.write(theDate.setMilliseconds(792)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setMinutes()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

date.setMinutes (minutes)

Description

The `setMinutes()` method sets the minutes in the `Date` object to the argument `minutes`, an integer from 0 to 59. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the minutes have been adjusted.

Example

The code in Listing 7.121 displays the current date in milliseconds before setting the minutes to 43. Notice that the milliseconds specified after the `setMinutes()` method are the same as the result returned from the method.

Listing 7.121 *Setting the Minutes with the* `setMinutes()` *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the minutes and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setMinutes returned ");
document.write(theDate.setMinutes(43)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setMonth()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`date.setMonth (month)`

Description

The `setMonth()` method sets the month in the `Date` object to the argument `month`, an integer from 0 (January) to 11 (December). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the month has been adjusted.

CAUTION

Prior to JavaScript 1.2, this method returned nothing.

Example

The code in Listing 7.122 displays the current date in milliseconds before setting the month to 2 (March). Notice that the milliseconds specified after the `setMonth()` method are the same as the result returned from the method.

Listing 7.122 Setting the Month with the `setMonth()` Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the month and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setMonth returned ");
document.write(theDate.setMonth(2)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>

```

Date.setSeconds()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.setSeconds(seconds)
```

Description

The `setSeconds()` method sets the seconds in the `Date` object to the argument `seconds`, an integer from 0 to 59. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the seconds have been adjusted.

CAUTION

Prior to JavaScript 1.2, this method returned nothing.

Example

The code in Listing 7.123 displays the current date in milliseconds before setting the seconds to 16. Notice that the milliseconds specified after the `setSeconds()` method are the same as the result returned from the method.

Listing 7.123 Setting the Seconds with the `setSeconds()` Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

```

Listing 7.123 Continued

```
//Set the seconds and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setSeconds returned ");
document.write(theDate.setSeconds(16)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setTime()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.setTime (milliseconds)
```

Description

The `setTime()` method sets the time in the `Date` object to the argument `milliseconds`, an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the desired date and time.

Example

The code in Listing 7.124 uses the `setTime()` method to set the date to November, 17, 2005 using milliseconds.

Listing 7.124 Setting the Date with the `setTime()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the date to Nov. 17, 2005.
theDate.setTime(1132203600000);

//Display the date.
document.write(theDate.toString());

// -->
</script>
```

Date.setUTCDate()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.setUTCDate(day)
```

Description

The `setUTCDate()` method sets the day of the month in the `Date` object to the argument day, an integer from 1 to 31 (universal time). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the day of the month has been adjusted.

Example

The code in Listing 7.125 displays the current date in milliseconds (universal time) before setting the day to 6. Notice that the milliseconds specified after the `setUTCDate()` method are the same as the result returned from the method.

Listing 7.125 *Setting the Date with the `setUTCDate()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the day of the month and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCDate returned ");
document.write(theDate.setUTCDate(6)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCFullYear()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
 Nav4+, IE 4+, Opera5+

Syntax

```
date.setUTCFullYear(year)
```

Description

The `setUTCFullYear()` method sets the year in the `Date` object to the argument year, a four-digit integer (universal time). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the year has been adjusted.

Example

The code in Listing 7.126 displays the current date in milliseconds (universal time) before setting the year to 2004. Notice that the milliseconds specified after the `setUTCFullYear()` method are the same as the result returned from the method.

Listing 7.126 *Setting the Year with the* `setUTCFullYear()` *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the year and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCFullYear returned ");
document.write(theDate.setUTCFullYear(2004)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCHours()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.setUTCHours(hours)
```

Description

The `setUTCHours()` method sets the hour in the `Date` object to the argument `hours`, an integer from 0 (12:00 a.m. midnight) to 23 (11:00 p.m.) universal time. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the hour has been adjusted.

Example

The code in Listing 7.127 displays the current date in milliseconds (universal time) before setting the hour to 5 (5:00 a.m.). Notice that the milliseconds specified after the `setUTCHours()` method are the same as the result returned from the method.

Listing 7.127 *Setting the Hour with the* `setUTCHours()` *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();
```

```
//Set the hour and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCHours returned ");
document.write(theDate.setUTCHours(5)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCMilliseconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.setUTCMilliseconds(milliseconds)
```

Description

The `setUTCMilliseconds()` method sets the milliseconds in the `Date` object to the argument `milliseconds`, an integer from 0 to 999 (universal time). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the milliseconds have been adjusted.

Example

The code in Listing 7.128 displays the current date in milliseconds (universal time) before setting the milliseconds to 258. Notice that the milliseconds specified after the `setUTCMilliseconds()` method are the same as the result returned from the method.

Listing 7.128 *Setting the Milliseconds with the* `setUTCMilliseconds()` *Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCMilliseconds returned ");
document.write(theDate.setUTCMilliseconds(258)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCMinutes()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

date.setUTCMinutes(minutes)

Description

The `setUTCMinutes()` method sets the minutes in the `Date` object to the argument `minutes`, an integer from 0 to 59 (universal time). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the minutes have been adjusted.

Example

The code in Listing 7.129 displays the current date in milliseconds (universal time) before setting the minutes to 19. Notice that the milliseconds specified after the `setUTCMinutes()` method are the same as the result returned from the method.

Listing 7.129 *Setting the Minutes with the `setUTCMinutes()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the minutes and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCMinutes returned ");
document.write(theDate.setUTCMinutes(19)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCMonth()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

date.setUTCMonth(month)

Description

The `setUTCMonth()` method sets the month in the `Date` object to the argument `month`, an integer from 0 (January) to 11 (December) universal time. The method returns an

integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the Date object after the month has been adjusted.

Example

The code in Listing 7.130 displays the current date in milliseconds (universal time) before setting the month to 4 (May). Notice that the milliseconds specified after the `setUTCMonth()` method are the same as the result returned from the method.

Listing 7.130 *Setting the Month with the `setUTCMonth()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the month and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCMonth returned ");
document.write(theDate.setUTCMonth(4)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setUTCSeconds()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera5+

Syntax

```
date.setUTCSeconds(seconds)
```

Description

The `setUTCSeconds()` method sets the seconds in the Date object to the argument `seconds`, an integer from 0 to 59 (universal time). The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the Date object after the seconds have been adjusted.

Example

The code in Listing 7.131 displays the current date in milliseconds (universal time) before setting the seconds to 46. Notice that the milliseconds specified after the `setUTCSeconds()` method are the same as the result returned from the method.

Listing 7.131 Setting the Seconds with the `setUTCSeconds()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the seconds and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setUTCSeconds returned ");
document.write(theDate.setUTCSeconds(46)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>
```

Date.setYear()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

`date.setYear(year)`

Description

The `setYear()` method sets the year in the `Date` object to the argument `year`. The argument can be either a four-digit or two-digit integer. To create a two-digit representation, subtract 1900 from the desired date. So if “92” was passed in as an argument then the year 1992 would be set. The method returns an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object after the year has been adjusted.

NOTE

The `getYear()` method is obsolete. The `getFullYear()` method should be used instead of `setYear()`.

CAUTION

Prior to JavaScript 1.2, this method returned nothing.

Example

The code in Listing 7.132 displays the current date in milliseconds before setting the year to 1983. Notice that the milliseconds specified after the `setYear()` method are the same as the result returned from the method.

Listing 7.132 Setting the Year with the `setYear()` Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Set the year and display the milliseconds.
document.write("Initial milliseconds=",theDate.getTime());
document.write("<br>setYear returned ");
document.write(theDate.setYear(83)," milliseconds.")
document.write("<br>Final milliseconds=",theDate.getTime());

// -->
</script>

```

Date.toGMTString()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera3+

Syntax

```
date.toGMTString()
```

Description

The `toGMTString()` method returns a string representing the universal time of the `Date` object. The date is converted to the GMT time zone before being converted to a string. The format of the string will look similar to the following:

```
Mon, 24 Oct 1982 12:03:27 GMT
```

Example

The code in listing 7.133 creates a `Date` object that contains the current date and time. The `toGMTString()` method returns the date as a string in GMT.

Listing 7.133 Converting Date and Time to GMT with the `toGMTString()` Method

```

<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Display date and time string in GMT.
document.write(theDate.toGMTString());

// -->
</script>

```

Date.toLocaleString()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, IE 3+, Opera5+

Syntax

```
date.toLocaleString()
```

Description

The `toLocaleString()` method returns a string representation of the `Date` object in the local time zone. The format of the string can vary greatly, depending on the user's date and time format settings.

Example

The code in Listing 7.134 creates a `Date` object that contains the current date and time. The `toLocaleString()` method returns the date and time as a string using the local time zone.

Listing 7.134 Converting Date and Time to the Local Time Zone with the `toLocaleString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Display date and time string.
document.write(theDate.toLocaleString());

// -->
</script>
```

Date.toSource()

JavaScript1.3, JScript3.0
Nav4.06+, IE4

Syntax

```
date.toSource()
```

Description

The `toSource()` method converts the `Date` object to a string that represents the source of the `Date` instance that was created.

NOTE

The `toSource()` method is usually called internally by JavaScript and not explicitly in code.

Example

Listing 7.135 creates a date object March 16, 2002. The `toSource()` method is then applied to the date object to return the source string `"(new Date(1018986503234))"` with the date expressed in milliseconds.

Listing 7.135 Accessing the Source of a Date Object with the `toSource()` Method

```
<script type="text/javascript1.3" language="JavaScript1.3">
<!--

//Create a Date object for April 16, 2002.
theDate = new Date(2002,3,16,15,48,23,234);

//Display "Tue Apr 16 15:48:23 GMT-0400 (Eastern Daylight Time) 2002".
document.write(theDate.toString());

//Display source of date object.
document.write(theDate.toSource()); //Display "(new Date(1018986503234))".

// -->
</script>
```

Date.toString()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE 3+, Opera5+

Syntax

```
date.toString()
```

Description

The `toString()` method returns a string representation of the Date object in the local time zone.

NOTE

How does the `toString()` method differ from `toGMTString()` and `toLocaleString()`? Unlike the `toGMTString()` method, the `toString()` method

displays the date in the local time zone. The `toString()` does not always display the date in the local format of the `toLocaleString()` method.

Example

The code in Listing 7.136 creates a `Date` object from the current date and time. The `toString()` method is then used to display the date as a string in the local time zone.

Listing 7.136 Accessing Date as a String with the `toString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Display date and time string.
document.write(theDate.toString());

// -->
</script>
```

Date.toUTCString()

JavaScript 1.2+, ECMAScript 1E+, JScript 3.0+
Nav4+, IE 4+, Opera3+

Syntax

```
date.toUTCString()
```

Description

The `toUTCString()` method returns a string representing the universal time of the `Date` object. The date is converted to the GMT time zone before being converted to a string. This method is exactly the same as the `toGMTString()`.

Example

The code in Listing 7.137 creates a `Date` object and initializes it to the current date and time. The `toUTCString()` method is then used to display the current date and time as a string in universal time.

Listing 7.137 Accessing Date in Universal Time with the `toUTCString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();
```

```
//Display date and time string in universal time.
document.write(theDate.toUTCString());

// -->
</script>
```

Date.unwatch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

Date.unwatch(property)

Description

The `unwatch()` method of the `Date` object is used to turn off the watch on a particular property specified by *property*.

Example

Listing 7.138 shows how the `unwatch()` method is used to stop watching the user-defined property `p`.

Listing 7.138 *Example of unwatch() Method of the Date Object*

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myDate." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myDate = new Date();
Date.prototype.p = 1;
myDate.watch("p",alertme);

myDate.p = 2;
myDate.unwatch("p");
myDate.p = 3;
myDate.watch("p",alertme);
myDate.p = 4;
// -->
</script>
```

Date.UTC()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE 3+, Opera3+

Syntax

Date.UTC(*year, month, day, hours, minutes, seconds, milliseconds*)

Description

Because the Date constructor creates new dates in local time, the UTC() method is provided to create dates in universal time (GMT). The method accepts arguments to allow you to set all aspects of a date and time, from the year to milliseconds. An integer, representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified, is returned from the method. The integer can then be used to create a new Date object. The arguments and return value are in Table 7.26.

CAUTION

The milliseconds returned from the method are incorrect in Navigator 2.

Table 7.26 Arguments and Return Value Associated with UTC()

Type	Item	Description
Argument	year	A four-digit representation of the year.
	month	The month represented as an integer where 0 represents January and 11 represents December.
	day	The day of the month represented as an integer from 1 to 31. Optional argument.
	hours	The hour represented as an integer where 0 represents 12 a.m. (midnight) and 23 represents 11 p.m. Optional argument.
	minutes	The minutes in the hour represented as an integer from 0 to 59. Optional argument.
	seconds	The seconds in the minute represented as an integer from 0 to 59. Optional argument.
	milliseconds	The milliseconds in the second represented as an integer from 0 to 999. Optional argument.
Returns		An integer, representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified, is returned from the method.

Example

The code in Listing 7.139 creates a new Date object initialized to September 29, 2002 (universal time). The newly created date and time are then displayed in the browser.

Listing 7.139 Creating a Date in Universal Time with the UTC() Method

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object in universal time.
theDate = new Date(Date.UTC(2002,9,29,20,5,8,10));

//Display date and time string.
document.write(theDate.toUTCString());

// -->
</script>
```

Date.valueOf()

JavaScript1.1+, ECMAScript 1E+, JScript3.0+
 Nav3+, NES2+, IE4+, Opera3+

Syntax

```
date.valueOf()
```

Description

The `valueOf()` method returns the primitive value of the object. In terms of an instance of a `Date` object, this method returns the equivalence of the `Date` object in milliseconds. The milliseconds are expressed as an integer representing the number of milliseconds between midnight January 1, 1970 (GMT) to the date and time specified in the `Date` object.

Example

Listing 7.140 creates a `Date` object representing the current date and time. The result of the `valueOf()` method on this date object is displayed in the browser.

Listing 7.140 Using the valueOf() Method to Return the Value of the Date Object

```
<script type="text/javascript" language="JavaScript">
<!--

//Create a Date object.
theDate = new Date();

//Display the source of the Dateobject.
document.write(theDate.valueOf());

// -->
</script>
```

Date.watch

JavaScript 1.2+
Nav4+, NES3+

Syntax

Date.watch (property, function)

Description

The `watch()` method of the `Date` object is used to turn on the watch on a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 7.141 shows how the `watch()` method is used to start watching the user-defined property `p`.

Listing 7.141 *Example of the `watch()` Method of the `Date` Object*

```
<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myDate." + id + " changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

var myDate = new Date();
Date.prototype.p = 1;
myDate.watch("p",alertme);

myDate.p = 2;
// -->
</script>
```

debugger

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+, Opera5+

Syntax

Reserved Keyword

Description

The keyword `debugger` is reserved for future use.

Example

This keyword has not been implemented; therefore no example is provided.

decodeURI()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
Nav6+, NES6+, IE5.5+

Syntax

`decodeURI (URIString)`

Description

The `decodeURI()` function is used to replace each escape sequence in the *URIString* (uniform resource identifiers) with the character that it represents based on the UTF-8 standard. The function returns the decoded string.

NOTE

Escape sequences that do not get encoded by the `encodeURI()` function (`,`, `/`, `?`, `:`, `@`, `&`, `=`, `+`, `$`) cannot be decoded using the `decodeURI()` function. To decode these reserved characters, use the `decodeURIComponent()` function.

Example

In Listing 7.142 the string "The apple is red." is encoded using the `encodeURI()` function. The result is the same string, but with the spaces replaced with "%20", the encoded equivalent. This encoded URI string is then decoded back to its original form using the `decodeURI()` function.

Listing 7.142 Using the `decodeURI()` Function

```
<script type="text/javascript" language='JavaScript'>
<!--

myString="The apple is red";
document.write("The original sentence: " + myString + "<br><br>");

myString = encodeURI(myString);
document.write("The encoded sentence: " + myString + "<br><br>");

myString = decodeURI(myString);
document.write("The decoded sentence: " + myString + "<br><br>");

// -->
</script>
```

decodeURIComponent()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
Nav6+, NES6+, IE5.5+

Syntax

`decodeURIComponent(URIComponentString)`

Description

The `decodeURIComponent()` function is used to replace each escape sequence in the *URIComponentString* (uniform resource identifiers) with the character that it represents based on the UTF-8 standard. The function returns the decoded string.

NOTE

The `decodeURIComponent()` function differs from the `decodeURI()` function in that it decodes some reserved characters (., /, ?, :, @, &, =, +, \$) that are not decoded by the `decodeURI()` function. For this reason, the `decodeURIComponent` method is designed to only decode URI strings that were originally encoded using the `encodeURIComponent()` function.

Example

In Listing 7.143 the phrase "fork, spoon, & knife" is encoded using the `encodeURIComponent()` function. The result is the same string but with the spaces, commas, and '&' characters encoded. This encoded URI component string is then decoded using the `decodeURI()` function. But notice that this function only decodes the spaces. In order to decode the commas and '&' character, the `decodeURIComponent()` function must be used.

Listing 7.143 Using the `decodeURIComponent()` Function

```
<script type="text/javascript" language='JavaScript'>
<!--

myString="fork, spoon, & knife";
document.write("The original phrase: " + myString + "<br><br>");

myString = encodeURIComponent(myString);
document.write("The encoded phrase: " + myString + "<br><br>");

myString = decodeURI(myString);
document.write("The decoded phrase: " + myString + "<br><br>");

myString = decodeURIComponent(myString);
document.write("The decoded component phrase: " + myString + "<br><br>");
```



```
// -->  
</script>
```

default

JavaScript1.2+, JScript3.0+
Nav4+, NES3+, IE4+, Opera5+

Syntax

Reserved Keyword

Description

The `default` keyword has not been implemented in JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented; therefore no example is provided.

delete

JavaScript 1.0+, ECMAScript 1E+, JScript 3.0+
Nav2+, NES3+, IE 4+, Opera3+

Syntax

`delete property`

Description

The `delete` operator deletes properties from objects and array elements from arrays by making them undefined. Actual memory deallocation is done by JavaScript garbage collection. Some objects, such as variables created by the `var` statement, are not affected by the `delete` operator.

CAUTION

Prior to JavaScript 1.2, the `delete` operator set the object property to `null` rather than undefined.

Example

The code in Listing 7.144 demonstrates how to use the `delete` operator to delete a `Date` object. Notice that the last `document.write()` statement does not write anything to the browser because `theDate` is undefined after using the `delete` operator.

Listing 7.144 Using the `delete` Operator

```
<script type="text/javascript" language="JavaScript">  
<!--
```

Listing 7.144 Continued

```
//Create a Date object in universal time.
theDate = new Date();

//Display the date object.
document.write("theDate=",theDate,"<br>Deleting theDate!<br>");

//Delete theDate.
delete theDate;

//Attempt to display theDate.
document.write("theDate=",theDate);    //theDate is undefined.

// -->
</script>
```

do

ECMAScript 3.0+, ECMAScript 2E+, JScript 5.5+
Nav4+, NES3+, IE 5.5+, Opera5+

Syntax

Reserved Keyword

Description

The do keyword is utilized in the do...while loop structure, thus it cannot be used as a variable name.

Example

This keyword is reserved. See the do...while entry for more information on its use.

do...while

JavaScript 1.0+ , ECMAScript 1E+, JScript 3.0+
Nav2+, NES3+, IE 4+, Opera 3+

Syntax

```
do{
    statement;
}while(expression);
```

Description

The do...while loop always executes the loop once before evaluating the expression for the first time. Once the loop has executed for the first time, the expression, in parenthesis, is evaluated. If true, the loop is executed again. When the expression

evaluates to `false`, the next line of code following the `while` structure is executed. A statement must be included in the loop that modifies a variable in the expression to prevent an infinite loop. Also, notice that a semicolon (;) must be placed after the right parenthesis.

Example

Listing 7.145 uses the `do...while` loop to control access through a gate. Only three individuals are allowed through the gate.

Listing 7.145 The `do...while` Loop

```
<script type="text/javascript" language="JavaScript">
<!--

names = new Array("Mendy","Michael","Jeff","Bill","Mike");
x = 0; //array counter

document.write("Allow these 3 individuals through gate:<br>");

//Print names of the first 3 individuals who can pass through gate.
do
{
    document.write((x+1),". ",names[x], "<br>");
    x++; //increment counter
}
while(x<3);

// -->
</script>
```

double

JavaScript1.2+, ECMAScript 2E+, JScript 5.0+
 Nav4+, NES3+, IE5+, Opera5+

Syntax

Reserved Keyword

Description

The `double` keyword has not been implemented in server-side JavaScript to date, but has been reserved for future use.

Example

This keyword has not been implemented; therefore no example is provided.

else

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, IE3+

Syntax

Reserved Keyword

Description

The `else` statement is used to perform alternative logic should the corresponding `if` statement not evaluate to `true`.

Example

Listing 7.146 shows an example of the basic logic to the `else` statement. If the `if` condition does not equate to `true`, the `else` statement is executed.

Listing 7.146 *Example of the `else` Keyword*

```
if (something == true)
{
    // Do something.
}
else{
    // Do something else.
}
```

encodeURIComponent()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
Nav6+, NES6+, IE5.5+

Syntax

`encodeURIComponent (URIString)`

Description

The `encodeURIComponent()` function is used to encode a *URI (uniform resource identifier)* by replacing specific characters with escape sequences representing UTF-8 encoding.

NOTE

Certain characters will not be encoded with the `encodeURIComponent()` function. They are shown in Table 7.27.

Table 7.27 *Unencoded Characters*

Type	Character
Reserved characters	, , / , ? , : , @ , & , = , + , \$
unescaped characters	alphabetic, decimal, - , . , ! , ~ , * , ' , (,)
score	#

Example

In Listing 7.147, the string "The apple is red." is encoded using the `encodeURIComponent()` function. The result is the same string, but with the spaces replaced with "%20", which is the encoded equivalent.

Listing 7.147 Using the `encodeURIComponent()` Function

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

myString="The apple is red";
document.write("The original sentence: " + myString + "<br><br>");

myString = encodeURIComponent(myString);
document.write("The encoded sentence: " + myString + "<br><br>");

// -->
</script>
</html>
```

`encodeURIComponent()`

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
 Nav6+, NES6+, IE5.5+

Syntax

`encodeURIComponent (URIComponentString)`

Description

The `encodeURIComponent()` function is used to replace certain characters in the *URIComponentString* (uniform resource identifier) with escape sequences representing its UTF-8 character encoding. This function assumes a URI component, and therefore will encode reserved characters.

NOTE

Certain characters will not be encoded with the `encodeURIComponent()` function. They are shown in Table 7.28.

Table 7.28 Unencoded Characters

Type	Character
unescaped characters	alphanumeric, decimal, -, ., !, ~, *, ', (,)
score	#

Example

In Listing 7.148, the phrase "fork, spoon, & knife" is encoded using the `encodeURIComponent()` function. The result is the same string, but with the spaces, commas, and & characters encoded.

Listing 7.148 *Using the `encodeURIComponent()` Function*

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

// Create a string object.
myString="fork, spoon, & knife";
document.write("The original phrase: " + myString + "<br><br>");

// URI Encode the string and display.
myString = encodeURIComponent(myString);
document.write("The encoded phrase: " + myString + "<br><br>");

// -->
</script>
</html>
```

enum

JavaScript 1.3+, ECMAScript 1E+, JScript 3.0+
Nav4.06+, IE4+

Syntax

Reserved Keyword

Description

The `enum` keyword is initially introduced in JavaScript 1.3 and is reserved for future use.

Example

This entry has not yet been implemented in the JavaScript language, and no example is provided.

Enumerator

JScript3.0+
IE4+

Syntax

```
var variable = new Enumerator(collection)
```

Description

The Enumerator object provides access to items in a collection by allowing iteration through a collection. Accessing items in a collection requires moving the Enumerator object to the first element or the next element using special methods. Unlike arrays, enumerators cannot access a specific position. To create an Enumerator object, use the `Enumerator()` constructor. This constructor requires that a collection be passed in as an argument. The methods associated with the Enumerator object are listed in Table 7.29.

Table 7.29 *Methods Associated with the Enumerator Object*

Method	Description
<code>atEnd()</code>	Determines whether Enumerator is at the end of the collection
<code>item()</code>	Returns the current item in the collection
<code>moveFirst()</code>	Resets the Enumerator to the first item in the collection
<code>moveNext()</code>	Moves Enumerator to the next item in the collection

Example

Listing 7.149 creates an Enumerator object for accessing items in the Drives collection.

Listing 7.149 *Creating an Enumerator Object*

```
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);
```

Enumerator.atEnd()

JScript 3.0+
IE4+

Syntax

```
enumeratorobj.atEnd()
```

Description

The `atEnd()` method of the Enumerator object returns `true` if the enumerator is pointing to the last element in the collection; otherwise, it returns `false`. `true` is also returned if the collection is empty or undefined.

Example

The `atEnd()` method is used in Listing 7.150 to determine when to stop looping through the `for` loop.

Listing 7.150 Stop Looping with the `atEnd()` Method

```
<html>

<h1>Drive Letters and drive types:</h1>

<script language="JScript">
<!--

// Create enumerator object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);

// Display drive letter and type for each drive.
// Loop on each Enumerator object. While not at the end of
// Enumeration, continue looping on the next item.
for (;!en.atEnd();en.moveNext())
{
    document.write("Drive ",en.item().DriveLetter);
    document.write(" is of type ",en.item().DriveType,"<br>");
}

// -->
</script>
</html>
```

Enumerator.item()

JScript 3.0+
IE4+

Syntax

```
enumeratorobj.item()
```

Description

The `item()` method of the `Enumerator` object returns the element to which the enumerator is pointing in the collection. If the collection is empty or undefined, `undefined` is returned.

Example

The `item()` method is used in Listing 7.151 to access the drive letter and drive type of the current drive.

Listing 7.151 Accessing Drives with the `item()` Method

```
<html>

<h1>Drive Letters and drive types:</h1>
```



```

<script language="JScript">
<!--

// Create enumerator object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);

// Display drive letter and type for each drive.
for (;!en.atEnd();en.moveNext())
{
    document.write("Drive ",en.item().DriveLetter);
    document.write(" is of type ",en.item().DriveType,"<br>");
}

// -->
</script>
</html>

```

Enumerator.moveFirst()

JScript 3.0+
IE4+

Syntax

```
enumeratorobj.moveFirst()
```

Description

The `moveFirst()` method of the `Enumerator` object moves the enumerator to the beginning of the collection. If the collection is empty or undefined, undefined is returned.

Example

In Listing 7.152, the enumerator is used to access the drive letters and types. The `moveFirst()` method moves the enumerator back to the beginning of the collection so that the drive status can be determined for each drive.

Listing 7.152 Moving the Enumerator to the Beginning of the Collection Using the `moveFirst()` Method

```

<html>

<h1>Drive Letters and drive types:</h1>

<script language="JScript">
<!--

// Create enumerator object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);

```

Listing 7.152 Continued

```
// Display drive letter and type for each drive.
for (;!en.atEnd();en.moveNext())
{
    document.write("Drive ",en.item().DriveLetter);
    document.write(" is of type ",en.item().DriveType,"<br>");
}

document.write("<h1>Drive Status:</h1>");

// Move enumerator to the beginning of the collection.
en.moveFirst();

// Determine if drive is ready.
for (;!en.atEnd();en.moveNext())
{
    if(en.item().IsReady)
    {
        document.write("Drive ",en.item().DriveLetter);
        document.write(" is ready!<br>");
    }
    else
    {
        document.write("Drive ",en.item().DriveLetter);
        document.write(" is not ready!<br>");
    }
}

// -->
</script>
</html>
```

Enumerator.moveNext()

JScript 3.0+
IE4+

Syntax

```
enumeratorobj.moveNext()
```

Description

The `moveNext()` method of the `Enumerator` object moves the enumerator to the next item in the collection. If the collection is empty, `undefined` is returned.

Example

The `moveNext()` method is used in Listing 7.153 to move the enumerator to the next drive in the collection.

Listing 7.153 Move to the Next Drive Using `moveNext()`

```

<html>

<h1>Drive Letters and drive types:</h1>

<script language="JScript">
<!--

// Create enumerator object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);

// Display drive letter and type for each drive.
for (;!en.atEnd();en.moveNext())
{
    document.write("Drive ",en.item().DriveLetter);
    document.write(" is of type ",en.item().DriveType,"<br>");
}

// -->
</script>
</html>

```

Error

JScript 5.0+
IE5+

Syntax

```

var variable = new Error();
var variable = new Error(num);
var variable = new Error(num, description);

```

Description

The Error object contains information about errors. The `Error()` constructors can be used to create custom error messages. The arguments and properties associated with the Error object are listed in Table 7.30.

Table 7.30 Arguments and Properties Associated with the Error **Object**

Type	Item	Description
Argument	number	A number assigned to an error—Zero, if no number is provided in the constructor.
	description	A string that describes the error—empty string, if no string is provided in the constructor.
Property	description	Sets or returns the description string associated with a specific error.

Table 7.30 Continued

Type	Item	Description
	message	A string that contains the error message.
	name	Returns a string containing the name of the error.
	number	Sets or returns the number associated with a specific error.

NOTE

An error number is a 32-bit value where the upper 16-bit word is the facility code and the lower word is the actual error code. The facility code refers to the component that caused the error.

Example

Listing 7.154 creates an Error object.

Listing 7.154 Creating an Error Object

```
var myError = Error(35, "My Error");
```

Error.description

JScript 5.0+
IE5+

Syntax

```
errorobj.description  
errorobj.description = string
```

Description

The description property associated with the Error object contains the description of the error. This method is read/write so that you can assign descriptions using this property.

Example

Listing 7.155 creates an Error object and then displays the description.

Listing 7.155 Display description property of an Error

```
<html>  
  
<script language="JScript">  
<!--  
  
// Create an Error object.  
var myError = new Error(45, "A really big error!");
```

```
// Display description associated with the custom error.
document.write("The custom error description is ");
document.write(myError.description, "");

// -->
</script>
</html>
```

Error.message

JScript 5.5+
IE5.5+

Syntax

```
errorobj.message
errorobj.message = string
```

Description

The message property associated with the Error object returns a string containing an error message displayed to users. This is the same as the *description* property.

Example

Listing 7.156 creates an Error object and then displays the message.

Listing 7.156 *Display the Message of an Error with the message Property*

```
<html>

<script language="JScript">
<!--

// Create an Error object.
var myError = new Error(45, "The error message!");

// Display message associated with the custom error.
document.write("The custom error message is ");
document.write(myError.message, "");

// -->
</script>
</html>
```

Error.name

JScript 5.5+
IE5.5+

Syntax

```
errorobj.name
```

Description

The name property associated with the Error object returns a name or exception type of an error. If the error is a runtime error, the name property is set to one of the exception types shown in Table 7.31.

Table 7.31 Runtime Exception Errors

Exception Type	Description
ConversionError	Occurs with invalid object conversions.
RangeError	Occurs when a function is supplied arguments outside its allowable range.
ReferenceError	Occurs when an invalid reference has been made.
RegExpError	Occurs with a regular expression compilation error.
SyntaxError	Occurs with invalid syntax.
TypeError	Occurs when actual type of an operand does not match expected type.
URIError	Occurs when an illegal URI is detected.

Example

Listing 7.157 creates an Error object and then displays the name property.

Listing 7.157 Displays the Name of an Error with the name Property

```
<html>

<script language="JScript">
<!--

// Create an Error object.
var myError = new Error(45,"The error message!");

// Display name associated with the custom error.
document.write("The error name is '");
document.write(myError.name,"'");

// -->
</script>
</html>
```

Error.number

JScript 5.0+
IE5+

Syntax

```
errorobj.number
errorobj.number = number;
```

Description

The number property associated with the Error object contains the error number of the error. This method is read/write, so you can assign an error number using this property.

Example

Listing 7.158 creates an Error object and then displays the error number.

Listing 7.158 Displays Error Number Associated with Custom Error Using the number Property

```
<html>

<script language="JScript">
<!--

// Create an Error object.
var myError = new Error(45, "A really big error!");

// Display error number associated with the custom error.
document.write("The custom error number is ");
document.write(myError.number);

// -->
</script>
</html>
```

escape()

JavaScript 1.0+, ECMAScript 1E+, JScript 3.0+
Nav2+, NES2+, IE4+, Opera3+

Syntax

```
escape (string)
escape(expression)
```

Description

The escape() function takes any string object or expression and returns a string object in which all non-alphanumeric characters are converted to their numeric equivalent in the form %XX. The XX is a hexadecimal number representing the non-alphanumeric character.

Example

Listing 7.159 shows how to use the escape() function. It takes a string of text and returns the escaped value.

Listing 7.159 Example of How to Use `escape()`

```
<script type="text/javascript" language="JavaScript">
var newString = escape("Tom & Jerry Show");
</script>
```

This returns: "Tom%20%26%20Jerry%20show". Notice that the spaces were replaced with "%20" and the & sign was replaced with %26.

Listing 7.160 shows how to return the `escape()` function's interpretation of user input.

Listing 7.160 Example of Returning a Value from the `escape()` **Function**

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Writes the value of input after escape has been performed.
function showEscapeVal(){
    alert("The escape value is: " + escape(document.form1.input1.value));
}
// -->
</script>

<form name="form1">
Enter input:
<input type="text" name="input1" size="30">
<input type="button" value="Show Escape Value" onClick='showEscapeVal()'>
</form>

</body>
</html>
```

eval()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`eval (command)`
`eval(string)`

Description

The `eval()` function accepts a string of JavaScript statements and evaluates it as JavaScript source code. `eval()` returns the value of the executed JavaScript statement.

NOTE

`eval()` has evolved with JavaScript. In the early releases of JavaScript, version 1.0, `eval()` was only a built-in function. When version 1.1 was released, though, it was made a method as well. But when JavaScript version 1.2 was initially released, it was changed back to being just a built-in function. However, `eval()` was once again changed to a built-in function and method with the release of Navigator 4.02 and later.

Example

Listing 7.161 shows how to use `eval()` to execute a JavaScript command input by a user. Simply enter a valid JavaScript command into the text box and click the Execute button. When the button is clicked, the `run()` function is called, which performs an `eval` on the user input.

Listing 7.161 Simple Use of the `eval()` Function

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Executes the JavaScript command entered in the text box.
function run(){
    eval (document.form1.input1.value);
}
// -->
</script>

<form name="form1">
Enter a Javascript command in the text field and click
the "execute" button to execute the command.
<br><br>
Command:<input type="text" name="input1" size="30">
<br>
<input type="button" value="execute" onClick='run()'>
</form>

</body>
</html>
```

export

JavaScript 1.2+, ECMAScript 1E+
Nav4+

Syntax

`export`

Description

`export` is a keyword used by scripts implementing security features that makes objects, properties, and methods accessible to other unsigned scripts.

Example

Listing 7.162 shows how to use the `export` keyword. In the JavaScript code, the `export` keyword is used to make the variables, `name`, `city`, and `showName()` function available to other unsigned scripts. Other scripts would access the information using the `import` keyword.

Listing 7.162 *Example of Using* `export`

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Declare variables.
var name = "John Smith";
var city = "Atlanta";

function showName(){
    alert("Your name is: " + name);
}

// Makes two variables and showName function available to other scripts.
export name, city, showName

// -->
</script>

</body>
</html>
```

extends

JavaScript 1.3+, ECMAScript 1E+
Nav4.06+

Syntax

Reserved Keyword

Description

The `extends` keyword was initially introduced in JavaScript 1.3, and it is reserved for future use.

Example

This entry has not yet been implemented in the JavaScript language.

false

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `false` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

final

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `final` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

float

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `float` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

for

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+

Syntax

```
for( [initial statement]; [condition]; [num];){  
    code;  
}
```

Description

The `for` keyword is used to create a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop.

The *initial statement* is a JavaScript statement or variable declaration. The *condition* is an optional statement that is evaluated through each iteration of the loop. If a *condition* is satisfied, all statements contained in the loop are executed. The *num* designates whether the loop increments or decrements every loop iteration. And finally, the *code* contains JavaScript statements that are executed each time the condition is satisfied.

Example

Listing 7.163 shows an example of a loop that performs a `document.write` nine times. An initial variable `i` is declared and set equal to zero. The variable is then evaluated against the condition that it is less than nine. If this evaluates to `true`, the variable is incremented and the `document.write` expression is performed.

Listing 7.163 Example of the `for` Loop

```
<html>  
<body>  
<script type="text/javascript" language="JavaScript">  
<!--  
  
    // Loop nine times and display text indicating each iteration.  
    for (i=0; i<9; i++){  
        document.write("Loop iteration " + i + "<br>");  
    }  
  
    // -->  
</script>  
  
</body>  
</html>
```

for...in

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
for(variable in object){  
  
    code;  
  
}
```

Description

The `for...in` object iterates a specified variable over all the properties of an object. The statements, contained in the body, are executed once for each property. The *variable* is a variable iterated over each property in the *object*. The *code* contains JavaScript statements to be executed.

Example

Listing 7.164 shows how to use the `for...in` object. The `showProperties` function takes an object type and object name as arguments. A `for...in` loop is then executed on the object displaying each of the object's properties and their values.

Listing 7.164 Using the `for...in` Object

```
<html>  
<body>  
<script type="text/javascript" language = "JavaScript">  
!--  
// Function displays the properties for the specified object.  
function showProperties(obj, objName) {  
    var result;  
    for (var i in obj) {  
        result += i + " = " + obj[i] + "<br>";  
    }  
    document.write("The properties for the " + objName + " object:" +  
        "<br><br>");  
    document.write(result);  
}  
// -->  
</script>  
<form name="form1">  
<input type="button" value="Get Button Properties" name="button1"  
onClick='showProperties(this, this.name)'  
</form>  
</body>  
</html>
```

function

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
function name(parm1, parm2, ..., parmN)
```

```
function name()
```

Description

The `function` keyword is used for specifying JavaScript functions. JavaScript functions typically contain a series of JavaScript statements that are grouped together to perform a specific task. The *name* is the name of the function and *parm1–parmN* are any optional parameters.

Example

Listing 7.165 shows how the `function` keyword is used. The `function` keyword defines the `sendMessage` function.

Listing 7.165 *Example of the function Keyword*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
function sendMessage(){
    alert("The function key word is used to declare the sendMessage
function");
}
// -->
</script>
</html>
```

Function()

JavaScript 1.0+, ECMAScript 1E+, JScript 2.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
var variable = new Function()
var variable = new Function(int)
var variable = new Function(arg1, ..., argN)
```

Description

`Function()` is a constructor that creates a `Function` object. Table 7.32 lists the different methods and properties of the `Function` object.

Table 7.32 *Properties and Methods of the Function Object*

Type	Item	Description
Property	arguments	Array reflecting function arguments
	arity	Number of arguments expected by function
	callee	Specifies the body of the current executing function
	caller	Reference to function caller
	constructor	Specifies the function which creates the Function object's prototype
	length	Returns the number of arguments expected by a function
Method	prototype	Prototype for a class of objects
	apply()	Applies method to multiple objects
	call()	Allows calling of methods belonging to other functions
	toSource()	Creates a copy of the function object
	toString()	Converts function back to string that defines it
	valueOf()	Returns a string representation of the function source code

Example

Listing 7.166 shows how a new Function object can be created with a single line. In this example, the function, when called, will change the background color of the page to blue.

Listing 7.166 *Setting the Background Color with a New Function Object*

```
<script type="text/javascript" language="JavaScript">
<!--

// Create a function to change background color.
var setBGColor = new Function(document.bgColor='blue');

// -->
</script>
```

Function.apply()

JavaScript 1.3+, ECMAScript 1E+
Nav4.06+, IE3+, Opera3+

Syntax

function.apply()

Description

The `apply()` method of the Function object is used to apply a method of one object to another object. Using the `apply()` method keeps developers from having to rewrite methods for different objects.

Example

Listing 7.167 shows how the `apply()` method can be used between two objects. The `bigHome` function adds a new property, `numRooms`, to the `home` object.

Listing 7.167 *Using the `apply()` Method*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function defines a home with two properties.
function home(number, street){
    var num = number;
    var str = street;
    document.write("house number is: " + num + "<br>");
    document.write("Street name is: " + str + "<br>");
}

// Function adds an additional property to the basic home.
// Applies the home function.
function bigHome(number, street, rooms){
    var numRooms = rooms;
    home.apply(home,arguments);
    document.write("The number of rooms is: " + numRooms + "<br>");
}

myHome = new bigHome(101, Main, 5);
// -->
</script>
</body>
</html>
```

Function.arguments

JavaScript 1.1+, JScript 2.0+, ECMAScript 1E+
Nav3+, NES2+, IE3+, Opera3+

Syntax

function.arguments

Description

The `arguments` property of the `Function` object is an array that holds the arguments that are passed to a function. The number of arguments passed to a defined function can be more than the number of parameters if the `arguments` array is used. The `arguments` array can only be accessed while inside the function. Any attempt to access the `arguments` array outside the function will result in an error.

Example

Listing 7.168 shows how the `arguments` array is used. A `document.write` is performed, which calls the `foo` function. The `foo` function is passed two arguments and calls the `foobar` function with the parameter 123. The `foobar` function outputs the number of arguments it has and outputs the second argument to the function that called it, in this case the `foo` function.

Listing 7.168 Using the `arguments` Property

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function foo, calls function foobar with the parameter 123.
function foo(a,b) {
    foobar(123);
    document.write("Done with function foo" + "<br>");
}

// Function foobar writes output to the document
// using the arguments property of function.
function foobar(x) {
    document.write(foobar.arguments.length + "<br>");
    document.write(foobar.arguments.caller.b + "<br>");
}

document.write(foo(21,44) + "\n");
// -->
</script>
</body>
</html>
```

Function.arity

JavaScript 1.2
 Nav4+, NES3+

Syntax

function.arity

Description

The `arity` property of the `Function` object represents the number of declared arguments a function expects to receive. This is valid when the `language` attribute of the `script` tag is set to `JavaScript1.2`. *This property is deprecated in JavaScript 1.4.* In JavaScript 1.5, this property is replaced with the `length` property.

Example

Listing 7.169 shows how `arity` can be used. The first line written to the user's page contains the number of arguments passed to the function. The second line is the result of running the script.

Listing 7.169 *Example of Using* `arity`

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function subtracts the second number from the first.
function subtract(first, second){
    var result = first - second;
    return result;
}

// Write the results to the screen.
document.write("arity = " + subtract.arity + "<br>")
document.write("The result of the subtract function is: " + subtract(4,3));

// -->
</script>
</body>
</html>
```

Function.call()

JavaScript 1.3+
Nav4.06+, IE3+, Opera3+

Syntax

```
function.call(this)
```

```
function.call(this, arg1, arg2, ..., argN)
```

Description

The `call()` method of the `Function` object allows you to call another object's method. Optional arguments can be passed to the method as shown in the second syntactical definition.

Example

Listing 7.170 shows an example of the `call()` method. The script creates a person and author object. The author object uses the `call()` method to perform some of its creation.

Listing 7.170 Using the `call()` Method of the Function Object

```
<script type="text/javascript" language="JavaScript">
<!--

// Create a person object to handle the creation
// of people.
function person (author, name){
    this.name = name;
    this.author = true;
}

// Create an author object.
function authors(name, books){
    this.books = books;
    person.call(this, name);
}

authors.prototype = new person();

// Create a new author.
var myAuthor = new authors("Allen", 5);

// -->
</script>
```

Function.callee

JavaScript 1.2+
Nav4+, IE5+

Syntax

function.callee

Description:

The `callee` property of the Function object was actually a property of `Function.arguments` in JavaScript versions through 1.3. In JavaScript 1.4 and later, it is a local property. It is used to specify the function body of the currently executing function. It is only available within the body of a function.

Example

Listing 7.171 shows how the `callee` property is used to get the value of the `Function.callee` property.

Listing 7.171 Accessing the *callee* Property

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
function display() {
    alert( display.callee );
}
// -->
</script>

<body>
<form name="form1">
<input type="button" value="Get callee property" onClick='display()'>
</form>

</body>
</html>
```

Function.caller

JavaScript 1.1+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

function.caller

Description:

The `caller` property of the `Function` object is used to reference the function that called the currently executing function. *This property is deprecated in JavaScript 1.3.* In JavaScript versions 1.4 and 1.5, this property is no longer used.

Example

Listing 7.172 shows how the `caller` property is used to get the name of the function calling `John()`.

Listing 7.172 Accessing the *caller* Property

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Define a simple function Alice(), which calls the John() function.
function Alice(){
```

```

        var Boss = true;
        John();
    }

    // Function outputs its caller.
    function John(){
        myBoss = John.caller.name;
        document.write("The boss is: " + myBoss + "<br>");
    }
    // -->
</script>

</html>

```

Function.constructor

JavaScript 1.1+, JScript 2.0+
 Nav3+, NES2+, IE4+

Syntax

function.constructor

Description

The constructor property of the Function object is used to create the object's prototype.

Example

Listing 7.173 shows how the constructor property is used. A function object named drive is defined. Then using the constructor property, we determine whether drive is a Function object.

Listing 7.173 Accessing the constructor Property

```

<html>
<script type="text/javascript" language="JavaScript">
<!--

function drive() {
    return "something";
}

if(drive.constructor == Function)
{
    document.write("Constructor matches");
}

// -->
</script>
</html>

```

Function.length

JavaScript 1.1+, ECMAScript 1E+
Nav3+

Syntax

function.length

Description:

The length property of the Function object is used to specify the number of arguments in which the function expects. This is not to be confused with the arguments.length property that provides the number of arguments actually passed into a function.

Example

Listing 7.174 shows how the length property is used.

Listing 7.174 Accessing the length Property

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
// Function to display the args.
function show(parm1, parm2) {
    alert( "Length is: " + length );
}
// -->
</script>

<body>
Click the button to get the number of arguments the show function expects
<br><br>
<form name="form1">
<input type="button" value="Get length property" onClick='show(1,2)'\>
</form>
</body>
</html>
```

Function.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

function.prototype.property

function.prototype.method

Description

The prototype property of the Function object refers to the object that serves as the prototype from which classes are created. prototype allows you to add new properties or methods to an existing class by adding them to the prototype associated with the constructor of that class.

Example

Listing 7.175 shows how the prototype property is used. The function setTask, which simply sets the task variable, is defined. Then a new prototype called duty is created for the String object. The duty prototype is set to call the setTask function. Now all String objects in the example have the setTask method.

Listing 7.175 Example for the prototype Property

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

var mytask = new String();

// sample function that sets the task string
function setTask(str){
    var task="Girls go shopping";

    if(str != null){
        task = str;
    }
    return task;
}

String.prototype.duty = setTask;

document.write("The first task is: " + mytask.duty("Nothing") + "<br>");
document.write("The next task is: " + mytask.duty());
// -->
</script>
</body>
</html>
```

Function.toSource()

JavaScript 1.3+
Nav4.06+, IE3+, Opera3+

Syntax

function.toSource()

Description

The `toSource()` method of the `Function` object allows you to create a copy of an object.

Example

Listing 7.176 uses the `toSource()` method on a newly created string. Running this script in a browser returns the following:

```
(new String("This is the source"))
```

Listing 7.176 *Using the `toSource()` Method*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Create a new String instance.
var aString = new String("This is the source");

// Call the toSource() method.
bString = aString.toSource();

// Write the returned value of calling the toSource()
// method to the page.
document.write(bString);

// -->
</script>

</body>
</html>
```

Function.toString()

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE4+, Opera3+

Syntax

```
function.toString()
```

Description

The `toString()` method of the `Function` object is used to convert a function to a text string. The method converts the function back to the JavaScript source that defines the function. The converted string includes all aspects of the defined function.

Example:

Listing 7.177 shows how the `toString()` method is used to convert a function to a text string.

Listing 7.177 Use of the `toString()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function just writes some text output.
function writeText(){
    document.write("Some dummy text");
}

// Call the toString() method.
var func = writeText.toString();

// Write the results to the page.
document.write("The string representation of the writeText");
document.write(" function looks like: " + "<br><br><b>");
document.write(func + "</b>");
// -->
</script>

</body>
</html>
```

Function.valueOf()

JavaScript 1.1+, JScript 2.0+
 Nav3+, IE4+, Opera5+

Syntax

function.valueOf()

Description

The `valueOf()` method of the `Function` object is used to obtain a string representation of the function source code. Typically this is a method used internally by the JavaScript interpreter and not explicitly in code. However it can be called directly if needed.

Example

Listing 7.178 shows how the `valueOf()` method is used. A function called `myTest` simply calls the `valueOf()` method to get the string representation of the `myTest` function.

Listing 7.178 Using the `valueOf()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

document.write("Below is the value returned from calling valueOf() on" +
" the myTest function." + "<br><br>");

// Execute function myTest.
document.write( "<b>" + myTest() );

function myTest()
{
    var temp;
    return (myTest.valueOf() );
}

// -->
</script>
</body>
</html>
```

GetObject

JScript 3.0+
IE4+

Syntax

`GetObject (path)`

`GetObject(path, app.type)`

Description

The `GetObject` function returns a reference to an Automation object from a file. The function can take up to two parameters. The *path* parameter represents the full path and name to the file containing the object you want to retrieve.

NOTE

Some of the applications you can retrieve objects from allow you to activate only part of the file. This can be achieved by placing an exclamation point, `!`, at the end of the *path* followed by the string that specifies the part you want to activate.

The *app.type* attribute is the program ID or formal definition of the object. The *app* portion represents the name of the application providing the object, and the *type* designates the type of object to create. If this parameter is not passed, the Automation object will try to determine which application to start.

NOTE

Use the `ActiveXObject` function if you want to create an instance without starting the object with a file loaded.

Example

Listing 7.179 shows the syntax that can be used to return an Automation object from a file. The second entry shows how you can activate only part of the object.

Listing 7.179 Using the `GetObject` Function

```
// Returning an object for this Excel file
var myObj = GetObject("C:\\TEMP\\TESTOBJ.XLS");

// Activating the first sheet in the file
var myObjRef = GetObject("C:\\TEMP\\TESTOBJ.XLS!sheet1");
```

Global

JavaScript 1.0+, JScript 3.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

Core JavaScript Object

Description

The `Global` object is a core object in the JavaScript language. Properties and functions that are not associated with any other object belong to this object. Table 7.33 shows the properties and methods of this object

Table 7.33 Properties and Functions of the `Global` Object

Type	Item	Description
Property	Infinity	Keyword that represents positive infinity
	NaN	Represents an object not equal to any number
	Number	Converts an object into a Number
	String	Converts an object into a String
	undefined	Represents an undefined value
Method	<code>decodeURI()</code>	Decodes a uniform resource identifier
	<code>decodeURIComponent()</code>	Decodes a uniform resource identifier encoded by the <code>encodeURIComponent()</code> method.
	<code>encodeURIComponent()</code>	Encodes a complete uniform resource identifier

Table 7.33 Continued

Type	Item	Description
	<code>encodeURIComponent()</code>	Encodes a uniform resource identifier component
	<code>escape()</code>	Returns a string object in which all non-alphanumeric characters are converted to their numeric equivalent
	<code>eval()</code>	Accepts a string of JavaScript statements and evaluates it as JavaScript source code
	<code>isFinite()</code>	Method used to determine whether a variable has finite bounds
	<code>isNaN()</code>	Method used to determine whether a variable is a valid number or not
	<code>parseFloat()</code>	Method used to convert a string to a number of type <code>float</code>
	<code>parseInt()</code>	Method used to convert a string to an integer
	<code>unescape()</code>	Method that takes a hexadecimal value and returns the ISO-Latin-1 ASCII equivalent

Example

See the separate entries for each of these properties and functions for examples. Note that the properties and functions are not referred to with a preceding `Global` reference, when being used.

Global.decodeURI()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
Nav6+, NES6+, IE5.5+

Syntax

`decodeURI (URIString)`

Description

The `decodeURI()` function is used to replace each escape sequence in the *URIString* (uniform resource identifier) with the character that it represents based on the UTF-8 standard. (See <http://www.utf-8.org/> for more information on UTF-8.) The function returns the decoded string.

NOTE

Escape sequences that do not get encoded by the `encodeURIComponent()` function (`,`, `/`, `?`, `:`, `@`, `&`, `=`, `+`, `$`) cannot be decoded using the `decodeURI()` function. To decode these reserved characters, use the `decodeURIComponent()` function.

Example

In Listing 7.180, the string "The apple is red." is encoded using the `encodeURIComponent()` function. The result is the same string but with the spaces replaced with "%20", the encoded equivalent. This encoded URI string is then decoded back to its original form using the `decodeURIComponent()` function.

Listing 7.180 Using the `decodeURIComponent()` Function

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

// Create a string.
myString="The apple is red";
document.write("The original sentence: " + myString + "<br><br>");

// Encode the string.
myString = encodeURIComponent(myString);
document.write("The encoded sentence: " + myString + "<br><br>");

// Decode the string.
myString = decodeURIComponent(myString);
document.write("The decoded sentence: " + myString + "<br><br>");

// -->
</script>
</html>
```

Global.decodeURIComponent()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
 Nav6+, NES6+, IE5.5+

Syntax

`decodeURIComponent (URIComponentString)`

Description

The `decodeURIComponent()` function is used to replace each escape sequence in the *URIComponentString* (uniform resource identifier) with the character that it represents based on the UTF-8 standard. The function returns the decoded string.

NOTE

The `decodeURIComponent()` function differs from the `decodeURI()` function in that it decodes some reserved characters (., /, ?, :, @, &, =, +, \$) that are not decoded by the `decodeURI()` function. For this reason, the `decodeURIComponent` method is designed to only decode URI strings that were originally encoded using the `encodeURIComponent()` function.

Example

In Listing 7.181, the phrase "fork, spoon, & knife" is encoded using the `encodeURIComponent()` function. The result is the same string, but with the spaces, commas, and & characters encoded. This encoded URI component string is then decoded using the `decodeURI()` function. But notice that this function only decodes the spaces. In order to decode the commas and the & character, the `decodeURIComponent()` function must be used.

Listing 7.181 Using the `decodeURIComponent()` Function

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

// Create a string object.
myString="fork, spoon, & knife";
document.write("The original phrase: " + myString + "<br><br>");

// Encode the string using the encodeURIComponent function.
myString = encodeURIComponent(myString);
document.write("The encoded phrase: " + myString + "<br><br>");

// Decode the string with the decodeURI function.
myString = decodeURI(myString);
document.write("The decoded phrase: " + myString + "<br><br>");

// Decode the string using the decodeURIComponent function.
myString = decodeURIComponent(myString);
document.write("The decoded component phrase: " + myString + "<br><br>");

// -->
</script>
</html>
```

Global.encodeURI()

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
Nav6+, NES6+, IE5.5+

Syntax

`encodeURI (URIString)`

Description

The `encodeURI()` function is used to encode a URI by replacing specific characters with escape sequences representing UTF-8 encoding.

NOTE

Certain characters will not be encoded with the `encodeURIComponent()` function. They are shown in Table 7.34.

Table 7.34 Unencoded characters

Type	Character
Reserved characters	, , / , ? , : , @ , & , = , + , \$
unescaped characters	alphabetic, decimal, -, ., !, ~, *, ' , (,)
score	#

Example

In Listing 7.182, the string "The apple is red." is encoded using the `encodeURIComponent()` function. The result is the same string, but with the spaces replaced with "%20", the encoded equivalent.

Listing 7.182 Using the `encodeURIComponent()` Function

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

// Create a string object.
myString="The apple is red";
document.write("The original sentence: " + myString + "<br><br>");

// Encode the string object and then output the result.
myString = encodeURIComponent(myString);
document.write("The encoded sentence: " + myString + "<br><br>");

// -->
</script>
</html>
```

Global.`encodeURIComponent()`

JavaScript 1.5+, ECMAScript 3E+, JScript 5.5+
 Nav6+, NES6+, IE5.5+

Syntax

`encodeURIComponent (URIComponentString)`

Description

The `encodeURIComponent()` function is used to replace certain characters in the *URIComponentString* (uniform resource identifier) with escape sequences representing

their UTF-8 character encoding. This function assumes a URI component and therefore will encode reserved characters.

NOTE

Certain characters will not be encoded with the `encodeURIComponent()` function. They are shown in Table 7.35.

Table 7.35 Unencoded Characters

Type	Character
unescaped characters	alphanumeric, decimal, <code>-, ., !, ~, *, ' , (,)</code>
score	<code>#</code>

Example

In Listing 7.183, the phrase "fork, spoon, & knife" is encoded using the `encodeURIComponent()` function. The result is the same string, but with the spaces, commas, and & characters encoded.

Listing 7.183 Using the `encodeURIComponent()` Function

```
<html>
<script type="text/javascript" language='JavaScript'>
<!--

// Create a new string object.
myString="fork, spoon, & knife";
document.write("The original phrase: " + myString + "<br><br>");

// Encode the string and output the result.
myString = encodeURIComponent(myString);
document.write("The encoded phrase: " + myString + "<br><br>");

// -->
</script>
</html>
```

Global.escape()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
escape (string)
escape(expression)
```

Description

The `escape()` method takes any string object or expression and returns a string object in which all non-alphanumeric characters are converted to their numeric equivalent in

the form %XX. The XX is a hexadecimal number representing the non-alphanumeric character.

Example

Listing 7.184 shows how to use the `escape()` method and what it returns. It takes a string of text and returns the escaped value.

Listing 7.184 *Example of How to Use* `escape()`

```
<script type="text/javascript" language="JavaScript">
var newString = escape("Tom & Jerry Show");
</script>
```

This returns: "Tom%20%26%20Jerry%20show". Notice that the spaces were replaced with "%20" and the & sign was replaced with %26.

Listing 7.185 shows how to return the `escape()` method's interpretation of user input.

Listing 7.185 *Example of Taking User Input and Passing Data to the* **Escape Method**

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Writes the value of input after escape has been performed.
function showEscapeVal(){
    document.write(escape(document.form1.input1.value));
}
// -->
</script>

<form name="form1">
<input type="text" name="input1" size=30>
</form>

</body>
</html>
```

Global.eval()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`eval (command)`
`eval(string)`

Description

The `eval()` function accepts a string of JavaScript statements and evaluates it as JavaScript source code. `eval()` returns the value of the executed JavaScript statement.

NOTE

`eval()` has evolved with JavaScript. In the early releases of JavaScript, version 1.0, `eval()` was only a built-in function. When version 1.1 was released though, it was made a method as well. But when JavaScript version 1.2 was initially released, it was changed back to being just a built-in function. However, `eval()` was once again changed to a built-on function and method, with the release of Navigator 4.02 and later. With JavaScript 1.4 and later, the `eval()` method is a top level function not associated with any object.

Example

Listing 7.186 shows how to use `eval()` to execute a JavaScript command input by a user. Simply enter a valid JavaScript command into the text box and click the Execute button. When the button is clicked, the `run()` function is called, which performs an `eval` on the user input.

Listing 7.186 *Simple Use of the `eval()` Function*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Executes the JavaScript command entered in the text box.
function run(){
    eval (document.form1.input1.value);
}
// -->
</script>

<form name="form1">
Enter a JavaScript command in the text field and click
the "execute" button to execute the command.
<br><br>
Command:<input type="text" name="input1" size="30">
<br>
<input type="button" value="execute" onClick='run()'>
</form>

</body>
</html>
```

Global.Infinity

JavaScript 1.3+, JScript 3.0+, ECMAScript 1E+
Nav4.06+, IE4+, Opera3+

Syntax

Infinity

Description

Infinity is a JavaScript keyword that represents positive infinity.

Example

Listing 7.187 shows how the Infinity keyword is used. An input text box is provided to enter a value to be compared to Infinity. If any number is entered, it will result in being less than Infinity. However, if the word Infinity is entered, it will result in being equal to Infinity.

Listing 7.187 *Example of Using the Infinity Property*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// function checks to see if the input is greater, less than, or equal
// to the value input by the user.
function checkNum(){

    input=document.form1.num.value;

    if(input < Infinity){
        alert("Your number is less than Infinity");
    }
    else if(input > Infinity){
        alert("Your number is greater than Infinity");
    }
    else if(input == Infinity){
        alert("Your number if equal to Infinity");
    }
}

// -->
</script>

<form name="form1">
Enter a number to compare against Infinity.
<br><br>
<input type="text" size="35" name="num">
```

Listing 7.187 Continued

```
<br><br>
<input type="button" value="Check Number" onClick='checkNum()'>
</form>
// -->
</script>

</body>
</html>
```

Global.isFinite()

JavaScript 1.3+, ECMAScript 1E+, JScript 3.0+
Nav4.06+, IE4+, Opera3+

Syntax

isFinite()

Description

The isFinite() method is used to determine whether a variable has finite bounds.

Example

In Listing 7.188, you see how the isFinite() method is used to verify whether the user input value has finite bounds.

Listing 7.188 Example of isFinite() Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Function to check whether the text entered
// is a number.
function checkNum(){
    var n = document.form1.text1.value;
    if(isFinite(n) == true){
        alert("Your entry had finite bounds");
    }
}
// -->
</script>

<form name="form1">
Enter a number or character into the text box and then click the check value
button to verify if the input is a number.
<br><br>
```

```

<input type="text" name="text1" size="3">
<br><br>
<input type="button" value="Check value" onClick='checkNum() '>
<br>
</form>

</body>
</html>

```

Global.isNaN()

JavaScript 1.1+, JScript 1.0+, ECMAScript 1E+
Nav3+, NES2+, IE3+, Opera3+

Syntax

isNaN (*variable*)

Description

The isNaN() function is used to determine whether *variable* is a valid number.

Example

In Listing 7.189, you see how the isNaN() function can be used to check user input.

Listing 7.189 Example of isNaN() Function

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function to check whether the user input
// is a valid number.
function checkNum(){
    var n = document.form1.text1.value;
    if(isNaN(n) == true){
        alert("Your entry is not a number");
    }
}
// -->
</script>

<form name="form1">
Enter a number or character into the text box and then click the check value
button to verify if the input is a number.
<br><br>
<input type="text" name="text1" size=3>
<br><br>
<input type="button" value="Check value" onClick='checkNum() '>

```

Listing 7.189 Continued

```
<br>
</form>

</body>
</html>
```

Global.NaN

JavaScript 1.3+, JScript 3.0+, ECMAScript 1E+
Nav4.06+, IE4+

Syntax

NaN

Description

The NaN object represents an object that is not equal to any number, including itself. NaN stands for *Not a Number*.

Example

Listing 7.190 shows how the NaN object is used in a comparison.

Listing 7.190 Example Using the NaN Object

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Checks to see if the string "a" is a number.
if ("a" != NaN){
    document.write("This is not a number");
}
// -->
</script>

</body>
</html>
```

Global.Number()

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

Number(obj)

Description

The `Number()` method takes whatever object is passed in as a parameter, *obj*, and converts it to a `Number`. If the object being converted does not contain a well-formed numeric literal, `NaN` is returned. When converting a `Date` object, the time is measured in milliseconds from Jan 1970 GMT.

Example

Listing 7.191 shows how the `Number()` method is used to convert a `Date` object into a number.

Listing 7.191 *Example Using the `Number()` Method*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Create a date object.
var mydate = new Date ("January 1, 2001");

// Convert the date to a Number object.
document.write("Converting the Date object to a number gives you: " +
Number(mydate) );
// -->
</script>

</html>
```

Global.parseFloat()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`parseFloat (string)`

Description

The `parseFloat()` method is used to convert a string to a number.

Example

Listing 7.192 shows how the `parseFloat()` method is used. In the example, `parseFloat` is called with two different strings. The first string, which contains numerical characters, is converted to a number without any problems. The second string, which contains alphabetic characters, is unable to be converted into a number.

Listing 7.192 Example of the `parseFloat()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Convert the "1245.31" string to a number.
document.write("The string 1245.31 converted is" +
↳parseFloat("1245.31") + "<br>");

// Try to convert the string "test" to a number.
// If not possible, then print error.
if( isNaN(parseFloat("test")) ){
    document.write("Cannot convert test string to a number.");
}
// -->
</script>

</body>
</html>
```

Global.parseInt()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`parseInt(string, radix)`

`parseInt(string)`

Description

The `parseInt()` method is used to convert a string to an integer. It can take *string* input with an optional *radix* input. The *radix* input represents the base of the number in the string.

Example

Listing 7.193 shows how `parseInt()` is used to parse a string. A few different examples are shown for different string types.

Listing 7.193 Example of the `parseInt()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Convert the "859" string to an integer.
```



```
document.write("The string 859 converted to an integer is: ");
document.write(parseInt("859") + "<br>");

// Converts a binary string into a integer.
document.write("The binary string 101101 converted to an integer is: ");
document.write(parseInt("101101", 2) + "<br>");

// Converts a hexadecimal string into an integer.
document.write("The hexadecimal string FA832B converted to an integer is: ");
document.write(parseInt("FA832B", 16) + "<br>");

// -->
</script>

</body>
</html>
```

Global.String()

JavaScript 1.2+
 Nav4+, NES3+, IE4+

Syntax

String(obj)

Description

The String() method takes whatever object is passed in as a parameter and converts it to a String. It is the same as using the toString() method.

Example

Listing 7.194 shows how the String() method is used to convert a Date into a String.

Listing 7.194 Example Using the String() Method

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Create a new Date object.
var myVar = new Date(473982503723);

// Convert the Date object to a String object.
document.write("Converting the Date object to a String gives you: " +
String(myVar) );

// -->
</script>
</html>
```

Global.unescape()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`unescape (string)`

Description

The `unescape()` method takes a hexadecimal value and returns the ISO-Latin-1 ASCII equivalent. This method performs the opposite operation of the `escape()` method and is commonly used to escape user-entered data before form submission.

Example

Listing 7.195 declares a local variable, `escapedVal`, and passes it to the `unescape()` method. The result, `"@"`, is then written to the page.

Listing 7.195 *Using the `unescape()` Method to Convert a Hexadecimal Value to Its ASCII Equivalent*

```
<script type="text/javascript" language="JavaScript">
<!--

// Create a variable.
var escapedVal = "%40";

// Evaluate the variable and place the value in a variable.
var unescapedVal = unescape(escapedVal);

document.write('The <i>escapedVal</i> value (' + escapedVal + ') ');
document.write("evaluates to " + unescapedVal);

// -->
</script>
```

Global.undefined

JavaScript 1.3+, JScript 5.5+
Nav4.06+, IE5.5+

Syntax

`undefined`

Description

The `undefined` property represents an undefined value.

Example

Listing 7.196 shows how the undefined property can be used to check whether a variable has been assigned.

Listing 7.196 *Example Using the* undefined *Property*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

var myVar;

// Check for variable initialization.
if (myVar == undefined){
    document.write("The variable, myVar has not been initialized!");
}

// -->
</script>

</html>
```

goto

JavaScript 1.3+
Nav4+

Syntax

Reserved Keyword

Description

The goto keyword has not been implemented in client-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

if

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
if(statement){
    code;
}
```

Description

The `if` statement is used to perform logic statements on specific cases. If *statement* evaluates to true, *code* is executed.

Example

In Listing 7.197, you see how the `if` statement is being used to determine whether the Tilly boolean value is true.

Listing 7.197 *Example of the `if` Statement*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

var Tilly = 1;

// Check to see if the variable is true.
if(Tilly == true){
    document.write("Statement is true");
}

// -->
</script>

</body>
</html>
```

if...else

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
if(statement){
    code;
}else(statement){
    code;
}
```

Description

The `if...else` statement is used to perform logic statements on specific cases. If *statement* is satisfied, its statements are executed. Otherwise, the *else statement* is evaluated. When *statement* evaluates to true, *code* is executed.

Example

In Listing 7.198, you see how the `if...else` statement is being used to determine whether the `isDog` boolean value is true.

Listing 7.198 Example of the `if...else` Statement

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

var isDog = 1;

// Check to see if Dog variable is true or false.
// Output message depending on the case.
if(isDog == true){
    document.write("Statement is true");
}
else{
    document.write("Statement is false");
}
// -->
</script>

</body>
</html>

```

implements

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `implements` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

import

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`import`

Description

The `import` keyword allows a script to import properties, functions, and objects from a signed script that has exported the information.

Example

Listing 7.199 shows how the `import` keyword is used to import multiple properties from another script.

Listing 7.199 *Example of the `import` Keyword*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Imports the variables name, city and state from another script.
// This makes those properties accessible to myObj.
import myObj.name;
import myObj.city;
import myObj.state;

// -->
</script>

</body>
</html>
```

in

JavaScript 1.4
Nav6+, IE5.5+

Syntax

Prop *in* objectName

Description

The `in` keyword will return `true` or `false` depending on whether a specified property, *prop*, is in a specified object, *objectName*. The property can be a string or numeric expression representing a property name or array index.

Example

Listing 7.200 shows how the `in` keyword is used to check to see whether the `length` property is in the `String` object.

Listing 7.200 *Example of the `in` Keyword*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Create a new String object.
var myStr = new String("Something");
```

```
// Check for "length".
if ("length" in myStr){
    document.write("length in myStr evaluates to true");
}

// -->
</script>
</body>
</html>
```

Infinity

JavaScript 1.3+, JScript 1.0+
Nav4.06+, IE3+, Opera3+

Syntax

Infinity

Description

Infinity is a JavaScript keyword that represents positive infinity.

Example

Listing 7.201 shows how the Infinity keyword is used. An input text box is provided to enter a value to be compared to Infinity. If any number is entered, it will result in being less than Infinity. However, if the word "Infinity" is entered, it will result in being equal to Infinity.

Listing 7.201 *Example of the infinity keyword*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function checks to see if the input is greater, less than, or equal
// to the value input by the user.
function checkNum(){

    input=document.form1.num.value;

    if(input < Infinity){
        alert("Your number is less than Infinity");
    }
    else if(input > Infinity){
        alert("Your number is greater than Infinity");
    }
    else if(input == Infinity){
        alert("Your number is equal to Infinity");
    }
}
```

Listing 7.201 Continued

```
}  
// -->  
</script>  
  
<form name="form1">  
Enter a number to compare against Infinity.  
<br><br>  
<input type="text" size="35" name="num">  
<br><br>  
<input type="button" value="Check Number" onClick='checkNum()'>  
</form>  
  
</body>  
</html>
```

instanceof

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The instanceof keyword is used to determine whether a specified object is of a specified object type. It will return true or false depending on this condition.

Example

Listing 7.202 shows how the instanceof keyword is used to check to see whether the length property is in the String object.

Listing 7.202 Example of the instanceof Keyword

```
<html>  
<body>  
<script type="text/javascript" language="JavaScript">  
<!--  
  
// Create a new String object.  
var myStr = new String("Tilly the Dog");  
  
// Check to see if the myStr object is an instance of the  
// String object.  
if (myStr instanceof String){  
    document.write("myStr is an instance of the String object");  
}
```



```
// -->  
</script>  
</body>  
</html>
```

int

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `int` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

interface

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `interface` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

isFinite()

JavaScript 1.3+, ECMAScript 1E+, JScript 3.0+
Nav4.06+, IE4+, Opera3+

Syntax

`isFinite()`

Description

The `isFinite()` method is used to determine whether a variable has finite bounds.

Example

In Listing 7.203, `isFinite()` method is used to verify whether the user input value has finite bounds.

Listing 7.203 *Example of the `isFinite()` Method*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function to check whether the input has finite bounds.
function checkNum(){
    var n = document.form1.text1.value;
    if(isFinite(n) == true){
        alert("Your entry has finite bounds");
    }
}

// -->
</script>

<form name="form1">
Enter a number or character into the text box and the click the check value
button to verify if the input has finite bounds.
<br><br>
<input type="text" name="text1" size="3">
<br><br>
<input type="button" value="Check value" onClick='checkNum()'>
<br>
</form>

</body>
</html>
```

isNaN()

JavaScript 1.1+, JScript 1.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

`isNaN (variable)`

Description

The `isNaN()` function is used to determine whether *variable* is a valid number.

Example

Listing 7.204, shows how the `isNaN()` function can be used to check user input.

Listing 7.204 Example of the isNaN() Function

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function to check to see if the user input is a valid number.
function checkNum(){
    var n = document.form1.text1.value;
    if(isNaN(n) == true){
        alert("Your entry is not a number");
    }
}
// -->
</script>

<form name="form1">
Enter a number or character into the text box and then click the check value
button to verify if the input is a number.
<br><br>
<input type="text" name="text1" size=3>
<br><br>
<input type="button" value="Check value" onClick='checkNum() '>
<br>
</form>

</body>
</html>

```

java

JavaScript 1.1+
 Nav3+, NES2+, Opera 3+

Syntax

```
java
```

Description

The java object allows you to access any class within the package java.*. It is shorter and works the same way as using the packages.java property.

Example

Listing 7.205 shows how the java property can be used to call the lang package and print out a string to the java console.

Listing 7.205 *Example Using the java Object*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Use the Java package to write text to the Java Console.
java.lang.System.out.println("Z Me Fly!");

// -->
</script>

</html>
```

javaArray

JavaScript 1.1+
Nav4+, NES2+

Syntax

```
var variable = arrayObject

var variable = package.newInstance(obj)
```

Description

The javaArray object represents a wrapped Java array that can be accessed from within a JavaScript program. In most situations, to create a javaArray type, simply store a java array datatype into a variable in JavaScript and it will automatically create the javaArray object. It can also be created by using the newInstance function and passing in the object to be created as the parameter. Table 7.36 shows the method and property of the javaArray object.

Table 7.36 *Property and Method of the javaArray Object*

Type	Item	Description
Property	length	Specifies the number of elements in the javaArray
Method	toString()	Returns a string representing the specified number object

Example

Listing 7.206 shows how a new javaArray object is created for the String object.

Listing 7.206 *Example of a javaArray Object*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// Create a new java String object.
var myStr = new java.lang.String("Example of javaArray");

// Create the new javaArray by assigning the array returned
// from getBytes to the variable myStrBytes.
var myStrBytes = myStr.getBytes();

document.write("The first byte contains: " + myStrBytes[0] + "<br>");
document.write("The second byte contains: " + myStrBytes[1] + "<br>");
document.write("The third byte contains: " + myStrBytes[2] + "<br>");

// -->
</script>

</html>
```

javaArray.length

JavaScript 1.1+

Nav3+, NES2+, Opera3+

Syntax

javaArrayName.length

Description

The length property of the javaArray object allows you to get the length of the javaArray.

Example

Listing 7.207 shows how the length property can be used to get the length of the *myStrBytes* javaArray object. Using a for loop, we loop through the whole length of the array and print out all its contents.

Listing 7.207 Example Using the javaArray length Property

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Create a new java String object.
var myStr = new java.lang.String("Example of javaArray");
var myStrBytes = myStr.getBytes();

document.write("The length of the array is: " + myStrBytes.length + "<br>");

for (i=0; i < myStrBytes.length; i++)
{
```

Listing 7.207 Continued

```
document.write("The value in myStrBytes index " + i + " is "
➡+ myStrBytes[i] + "<br>");
}

// -->
</script>

</html>
```

javaArray.toString

JavaScript 1.1+
Nav3+, NES2+, Opera3+

Syntax

javaArrayName.toString()

Description

The `toString()` method of the `javaArray` object returns the string representation of the `javaArray`. In JavaScript 1.4, this method is overridden by `java.lang.Object.toString`.

Example

Listing 7.208 shows how the `toString()` method is used to output the string representation of the `myStrBytes`.

Listing 7.208 Example Using the toString Method

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Create a new java String object.
var myStr = new java.lang.String("Example of javaArray");
var myStrBytes = myStr.getBytes();

document.write("The toString representation of the array is: "
➡+ myStrBytes.toString() + "<br>");

// -->
</script>

</html>
```

javaClass

JavaScript 1.1+
Nav3+, NES2+, Opera3+

Syntax

```
javaClass = classname
```

Description

The `javaClass` object provides you a reference to a single class in the Java package. A `javaClass` object is created by assigning a specific class object to a JavaScript defined variable. Once created, the JavaScript defined variable can access static fields and static methods of the instantiated class.

Example

Listing 7.209 shows how the `javaClass` property is used to reference the `Color` class within the `java.awt` package. After creating the `javaClass` object, `col`, a new color is created using the specified RGB values.

Listing 7.209 Example Using the `javaClass` Object

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

col = package.java.awt.Color

// Create color object with RGB values (Maroon).
myColor = col(80,00,00);

// -->
</script>

</body>
</html>
```

javaObject

JavaScript 1.1+
Nav3+, NES2+, Opera3+

Syntax

```
javaObject = new Packages.JavaClass(parms)
```

Description

The `javaObject` object is a wrapped Java object that can be accessed within JavaScript code. It can be created in a script by assigning the return value of any Java method that returns an object type or by creating a new Java object.

Example

Listing 7.210 shows how the `javaObject` is used to get the `byteValue` of a Java Integer object.

Listing 7.210 Example Using the javaobject Object

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new java Integer object.
myInteger = new java.lang.Integer(492);
myByteVal = myInteger.byteValue();

// Display a specific byteValue.
document.write("This is the byteValue of 492: " + myByteVal);

// -->
</script>

</html>
```

javaPackage

JavaScript 1.1+
Nav3+, NES2+, Opera3+

Syntax

Packages.javaPackage

Description

The javaPackage object provides you with a reference to a specified Java Package.

Example

Listing 7.211 shows how the javaPackage is used to access the Java Font package and create a font object that has the UNDERLINE_ON property.

Listing 7.211 Example Using the javaPackage Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

// Specify the font java package.
var myAWT = Packages.java.awt.font;

// Create a new Font object.
myfont = myAWT.TextAttribute.UNDERLINE_ON;

document.write("My Font is: " + myfont);
```



```
// -->
</script>

</html>
```

JSEException

JavaScript 1.1+
Nav4+, NES2+

Syntax

- a) `public JSEException()`
- b) `public JSEException( String s)`
- c) `public JSEException (String s, String filename, int lineNum,
↳String source, int token)`

Description

The `JSEException` object is an exception that is thrown when a piece of JavaScript code returns an error. It is part of the `netscape.javascript` package. The `s` parameter represents the message string. The `filename` parameter represents the URL of the file containing the error. The `lineNum` parameter represents the line number in the file in which the error occurred. The `source` parameter represents the string containing the code to be executed. The `token` parameter represents the index into the source string where the error occurred. The constructors listed previously are deprecated in JavaScript 1.4. Table 7.37 shows the `JSEException` method.

Table 7.37 Method of the `JSEException` Object

Method	Description
<code>getWrappedException()</code>	Returns the exception thrown

Example

Listing 7.212 shows how a `JSEException` is used within a `try...catch` block. When running the `eval()` method, if the `dog` object is not defined, an exception will be thrown. When the catch block catches the exception, it will check to see whether the exception is a `JSEException`. If it is, `doRoutineA` is called. Otherwise `doRoutineB` is called.

Listing 7.212 Example of a `JSEException` Object

```
try {
    // Execute the specified statement.
    global.eval("dog.weight = 120;");
} catch (Exception e) {
    // If a JSEException is caught, execute doRoutineA.
```

Listing 7.212 Continued

```
    if (e instanceof JSEException) {
        doRoutineA();
    } else {
        doRoutineB();
    }
}
```

JSEException.getWrappedException()

JavaScript 1.4+
Nav6+, IE5.5+

Syntax

JSEException.getWrappedException()

Description

The `getWrappedException()` method is used to unwrap a `JSEException`. Typically, when JavaScript throws an exception, it is wrapped as an instance of `JSEException`.

Example

Listing 7.213 shows how the `getWrappedException()` method is used to return the exception thrown as an unwrapped exception.

Listing 7.213 Example Using the `getWrappedException()` Method

```
import netscape.javascript.*;

public class test {
    public static Object execute(JSObject obj, String jsCode) {
        try {
            obj.eval(jsCode);
        }
        catch (JSEException e) {
            // If a JSEException is caught, then wrap and return.
            return e.getWrappedException();
        }
        return null;
    }
}
```

JSObject

JavaScript 1.1+
Nav4+, NES2+

Syntax

JSObject

Description

The `JSObject` object is a wrapped instance of the `netscape.javascript.JSObject` class. This wrapped instance allows a Java program to manipulate a JavaScript object. Table 7.38 lists the property and methods associated with the `JSObject`.

Table 7.38 *Property and Methods of the JSObject Object*

Type	Item	Description
Property	<code>length</code>	Specifies the number of elements in the <code>javaArray</code> .
Method	<code>call()</code>	Calls a JavaScript method.
	<code>equals()</code>	Checks whether two <code>JSObjects</code> refer to the same instance.
	<code>eval()</code>	Evaluates a JavaScript expression.
	<code>getMember()</code>	Retrieves a JavaScript property value.
	<code>getSlot()</code>	Retrieves a JavaScript array element value.
	<code>getWindow()</code>	Gets a <code>JSObject</code> for the window containing the applet.
	<code>removeMember()</code>	Removes a property of a JavaScript object.
	<code>setMember()</code>	Sets a JavaScript object property value.
	<code>setSlot()</code>	Sets a JavaScript object array element value.
	<code>toString()</code>	Converts a <code>JSObject</code> to a string.

Example

Listing 7.214 shows how a new `JSObject` object is used to access the properties of the JavaScript `Car` object. In Listing 7.215, the JavaScript code to create a `Car` object is displayed.

Listing 7.214 *Example of the JSObject Object*

```
import netscape.javascript.*;

public class car
{
    public String carModel;
    public String carColor;
    public String carYear;

    // Define the class constructor.
    public car(JSObject jsCar)
    {
        this.carModel = (String)jsCar.getMember("model");
        this.carColor = (String)jsCar.getMember("color");
        this.carYear = (String)jsCar.getMember("year");
    }
}
```

Listing 7.215 Creating the Car Object

```
<html>
<!--
<script type="text/javascript" language="JavaScript">

function Car(model,color,year) {
    this.model = model
    this.color = color
    this.year = year
}

// Create a new Car object.
myCar = new Car("M3","black","2001");

</script>
// -->

</html>
```

JSObject.call()

JavaScript 1.1+
Nav3+, NES2+

Syntax

JSObject.call(methodName, argArray)

Description

The `call()` method allows you to invoke a JavaScript method from within a Java program. Pass it the *methodName* and an array representing an array of Java objects used to pass arguments to the JavaScript method.

Example

Listing 7.216 shows how the `call()` method is used to inform the user of the browser name and version. When the applet is loaded in the browser, the `init()` method contains a call to the JavaScript's test function, which provides the alert. Listing 7.217 shows the HTML code for loading the applet.

Listing 7.216 Java Applet Showing the call Method

```
import java.applet.*;
import netscape.javascript.*;
import java.awt.Graphics;

public class myApplet extends Applet {
    String myString;
```

```
// Initialize the applet.
public void init() {
    myString = new String("Pure JavaScript!");
    JSObject win = JSObject.getWindow(this);
    String args[] = {" "};
    win.call("test", args);
}

public void paint(Graphics g) {
    g.drawString(myString, 25, 20);
}

// Set the myString variable.
public void setString(String aString) {
    myString = aString;
    repaint();
}
}
```

Listing 7.217 Example of a JavaScript Program for the `call` Method

```
<html>
<script type="text/javascript" language="JavaScript">
<!--
// Function to check the version of the browser being used.
function test() {
    alert("You are using " + navigator.appName + " " + navigator.appVersion)
}
// -->
</script>

<body>
<applet code="myApplet.class" name="myApplet" width=150 height=25 mayscript>
</applet>

<form name="form1">
<input type="button" value="Set String"
  ↪onClick="document.myApplet.setString(document.form1.str.value)"><br>
<input type="text" size="20" name="str">
</form>
</body>
</html>
```

JSObject.equals()

JavaScript 1.1+
Nav3+, NES2+

Syntax

`JSObject.equals()`

Description

The `equals()` method is used to check whether two `JSObjects` refer to the same instance.

Example

Listing 7.218 shows how the `equals()` method is used to see whether the *win* object and the *joe* object are equal.

Listing 7.218 *Example Using the equals Method*

```
// Function to check whether two objects are equal.
public void check() {
    win = JSObject.getWindow(this);
    joe = JSObject.getMember("address");

    if(joe.equals(win) ){
        System.out.println("Objects are equal");
    }
    else {
        System.out.println("Objects are NOT equal");
    }
}
```

JSObject.eval()

JavaScript 1.1+
Nav3+, NES2+

Syntax

`JSObject.eval()`

Description

The `eval()` method is used to execute a JavaScript expression.

Example

Listing 7.219 shows how the `eval()` method is used in the Java program to call an `alert()` JavaScript function indicating that the Applet is painting. Listing 7.220 shows the HTML for loading the Applet.

Listing 7.219 *Example of using the eval() Method Within Java*

```
import java.applet.*;
import netscape.javascript.*;
import java.awt.Graphics;
```

```

public class myApplet extends Applet {
    String myString;

    // Initialize the applet.
    public void init() {
        myString = new String("Pure JavaScript!");
    }

    public void paint(Graphics g) {
        g.drawString(myString, 25, 20);
        JSObject win = JSObject.getWindow(this);
        win.eval("alert('Painting')");
    }

    // Set the myString variable.
    public void setString(String aString) {
        myString = aString;
        repaint();
    }
}

```

Listing 7.220 Example of JavaScript Loading Applet

```

<html>
<body>

<!-- Load the applet in the browser -->
<applet code="myApplet.class" name="myApplet" width=150 height=25 mayscript>
</applet>

<form name="form1">
<input type="button" value="Set String"
onClick="document.myApplet.setString(document.form1.str.value)"><br>
<input type="text" size="20" name="str">
</form>

</body>
</html>

```

JSObject.getMember()

JavaScript 1.1+
 Nav3+, NES2+

Syntax

JSObject.getMember()

Description

The `getMember()` method is used to retrieve the value of a property of a JavaScript object.

Example

Listing 7.221 shows how the `getMember()` method is used to get the properties of the new car object created from the JavaScript script.

Listing 7.222 shows the JavaScript code for creating a new Car object.

Listing 7.221 Example Using the `getMember()` Method to Get the Properties of the Car Object

```
import netscape.javascript.*;

public class car
{
    public String carModel;
    public String carColor;
    public String carYear;

    // Define the class constructor.
    public car(JSObject jsCar)
    {
        this.carModel = (String)jsCar.getMember("model");
        this.carColor = (String)jsCar.getMember("color");
        this.carYear = (String)jsCar.getMember("year");
    }
}
```

Listing 7.222 Example of Creating a New Car Object

```
<html>
<!--
<script type="text/javascript" language="JavaScript">

function Car(model,color,year) {
    this.model = model
    this.color = color
    this.year = year
}

// Create a new Car object.
myCar = new Car("M3","black","2001");

</script>
// -->

</html>
```


JSObject.getSlot()

JavaScript 1.1+
Nav3+, NES2+

Syntax

JSObject.getSlot()

Description

The `getSlot()` method is used to get the value of an array element of a JavaScript object.

Example

Listing 7.223 shows how the `getSlot()` method is used to get the value of the second array element of a `Car` object.

Listing 7.223 Example Using the `getSlot()` Method

```
import netscape.javascript.*;

public class car
{
    public car(JSObject jsCar)
    {
        // Get the second array element.
        jsCar.getSlot(2);
    }
}
```

JSObject.getWindow()

JavaScript 1.1+
Nav3+, NES2+

Syntax

JSObject.getWindow()

Description

The `getWindow()` method is a static method that returns a handle to the current Navigator window.

Example

Listing 7.224 shows how the `getWindow()` method is used to assign a window handle to the `win` `JSObject`.

Listing 7.224 Example Using the `getWindow()` Method

```
public class myApplet extends Applet {
    public void init() {
        JSObject win = JSObject.getWindow(this);
    }
}
```

JSObject.removeMember()

JavaScript 1.1+
Nav3+, NES2+

Syntax

```
JSObject.removeMember()
```

Description

The `removeMember()` method is used to remove a property of a JavaScript object.

Example

Listing 7.225 shows how the `removeMember()` method is used to remove the *year* property of the *car* object.

Listing 7.226 shows the JavaScript code for creating a new *car* object.

Listing 7.225 Example Using the `removeMember()` Method to Remove the *year* Property

```
import netscape.javascript.*;

public class car
{
    // Define the class constructor.
    public car(JSObject jsCar)
    {
        jsCar.removeMember("year");
    }
}
```

Listing 7.226 Example Defining the *Car* Object

```
<html>
<!--
<script type="text/javascript" language="JavaScript">

function Car(model,color,year) {
    this.model = model
    this.color = color
    this.year = year
}
```

```
// Create a new Car object.  
myCar = new Car("M3","black","2001");  
  
</script>  
// -->  
  
</html>
```

JSONObject.setMember()

JavaScript 1.1+
Nav3+, NES2+

Syntax

JSONObject.setMember()

Description

The `setMember()` method is used to set the value of a property of a JavaScript object.

Example

Listing 7.227 shows how the `setMember()` method is used to set the value of the `model` property in the JavaScript `car` object (defined in Listing 7.xxx).

Listing 7.227 Example Using the `setMember()` Method to Set the `model` Property

```
import netscape.javascript.*;  
  
public class car  
{  
    public car(JSONObject jsCar)  
    {  
        // Set value of Model property  
        jsCar.setMember("model","330ci");  
    }  
}
```

JSONObject.setSlot()

JavaScript 1.1+
Nav3+, NES2+

Syntax

JSONObject.setSlot()

Description

The `setSlot()` method is used to set the value of an array element of a JavaScript object.

Example

Listing 7.228 shows how the `setSlot()` method is used to set the value of the second array element to the string `0-60`.

Listing 7.228 *Example Using the `setSlot()` Method*

```
import netscape.javascript.*;

public class car
{
    public car(JSObject jsCar)
    {
        // Set the value of array element 2.
        jsCar.setSlot(2, "0-60");
    }
}
```

JSObject.toString()

JavaScript 1.1+
Nav3+, NES2+

Syntax

`JSObject.toString()`

Description

The `toString()` method is used to obtain the string representation of the `JSObject`.

Example

Listing 7.229 shows how the `toString()` method is used to get the string representation of the `win` object.

Listing 7.229 *Example Using the `toString()` Method*

```
public void display() {
    win = JSObject.getWindow(this);
    win.eval("alert(win.toString() )" );
}
```

label

JavaScript 1.2+, JScript 1.0+
Nav4+, NES3+, IE3+

Syntax

`label:`

`code;`

Description

The *label* keyword provides an identifier that can be used with *break* or *continue* to indicate where a program should continue execution. You can associate a block of JavaScript statements with each other using a label. When a label is called using *break* or *continue*, *code* is executed.

Example

Listing 7.230 shows an example of the *label* statement in conjunction with *break*.

Listing 7.230 Example of *label*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Define a label called doSomething.
doSomething:
    var x = 2+2;
    var y = 5*5;
    document.write("The value of 2+2 is: " + x + "<br>");

    // Define another label called doSomeMore.
    doSomeMore:
        for(i=0; i<5; i++){
            document.write("Ths value of i is: " + i + "<br>");
            if(i==2){
                break doSomeMore;
            }
        }
        document.write("I did some more." + "<br>");

    document.write("The value of 5 x 5 is: " + y + "<br>");
// -->
</script>

</body>
</html>
```

long

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The long keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

Math()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

Core JavaScript Object

Description

The Math object is a built-in object containing properties and methods used for mathematical computation. It is a predefined JavaScript object and can be accessed without the use of a constructor or calling method. All Math properties and methods are static. Table 7.39 shows the different methods and properties of the Math object.

Table 7.39 Properties and Methods of the Math Object

Type	Item	Description
Property	ceil	Returns the smallest integer greater than or equal to a number
	E	Returns the value for Euler’s constant
	LN10	Returns the natural logarithm of 10
	LN2	Returns the natural logarithm of 2
	LOG10E	Returnsthe base 10 logarithm of E
	LOG2E	Returns the base 2 logarithm of E
	PI	Returns the value of PI
	SQRT1_2	Returns the square root of 1/2
	SQRT2	Returns the square root of 2
Method	abs()	Returns the absolute value of a number
	acos()	Returns the arccosine of a number
	asin()	Returns the arcsine of a number
	atan()	Returnsthe arctangent of a number
	atan2()	Returns the arctangent of the quotient of its paramet-ers
	cos()	Returns the cosine of a number
	exp()	Returns E ^x , where x is a number
	floor()	Returns the largest integer less than or equal to a number
	log()	Returns the natural logarithm (base E) of a number

Type	Item	Description
	<code>max()</code>	Returns the larger of two arguments
	<code>min()</code>	Returns the smaller of two arguments
	<code>pow()</code>	Returns base to the exponent power, base^{exp}
	<code>random()</code>	Returns a random number between 0 and 1
	<code>round()</code>	Rounds a number to its nearest integer
	<code>sin()</code>	Returns the sine of a number
	<code>sqrt()</code>	Returns the square root of a number
	<code>tan()</code>	Returns the tangent of a number
	<code>toSource()</code>	Creates a copy of an object
	<code>toString()</code>	Returns a string representation of an object
	<code>unwatch()</code>	Removes a watchpoint
	<code>watch()</code>	Sets a watchpoint

Example

Listing 7.231 shows how to create a new Math object.

Listing 7.231 Example of Creating a Math Object

```
<html>
<body>
<title>Example of creating a Math object</title>
<script type="text/javascript" language="JavaScript">
<!--
// Create a new Math object.
var newMathObject = Math.E;
// -->
</script>

</body>
</html>
```

Math.abs()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.abs (num)`

Description

The `abs()` method of the Math object is used to calculate the absolute value of *num* compared to the Math object on which it is invoked.

Example

Listing 7.232 shows how to use the `abs()` method and what it returns.

Listing 7.232 Example of the `abs()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the absolute value of the input number.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.abs(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the absolute value of the number entered.
<br><br>
Enter Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
Answer:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.acos()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.acos(num)`

Description

The `acos()` method of the `Math` object is used to calculate the arccosine of a number. The return value is between 0 and π and is measured in radians. If the return value is outside this range, 0 is returned.

Example

Listing 7.233 shows how the `acos()` method is used and what values can be returned.

Listing 7.233 Example of How to Use `acos()`

```
<html>
<body>
```



```

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the arccosine of the input number.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.acos(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the arccosine of the number entered.
<br><br>
Enter Number:
<input type="text" name="input" size=10>
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The arccosine is:
<input type="text" name="answer" size=10>
</form>

</body>
</html>

```

Math.asin()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.asin (num)

Description

The `asin()` method of the `Math` object calculates and returns the arcsine of a number. The return value is $-\pi/2$ and $\pi/2$ in radians. If the return value is not within this range, `0` is returned.

Example

Listing 7.234 shows how the `asin()` method is used.

Listing 7.234 Example of `asin()`

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the arcsine of the input number.
function doMath(){

```

Listing 7.234 Continued

```
var inputNum=document.form1.input.value;
var result = Math.asin(inputNum);
document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the arcsine of the entered number.
<br><br>
Enter Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The arcsine is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.atan()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.atan (num)

Description

The `atan()` method of the `Math` object is used to calculate the arctangent of a number. The return value is a numeric value between $-\pi/2$ and $\pi/2$ radians.

Example

Listing 7.235 shows how the `atan()` method is used to calculate the arctangent of a number.

Listing 7.235 Example of the `atan()` Method

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the arctangent of the input number.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.atan(inputNum);
```

```

    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the arctangent of the input number.
<br><br>
Enter Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The arctan is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>

```

Math.atan2()

JavaScript 1.0+, ECMAScript 1E+, JScript 3.0+
 Nav2+, NES2+, IE4+

Syntax

```
math.atan2(num1, num2)
```

Description

The `atan2()` method of the `Math` object is used to calculate the arctangent of the quotient of its parameters. It returns a numeric value between $-\pi$ and π representing the angle theta of an (x, y) point.

Example

Listing 7.236 shows an example of how the `atan2()` method is used. The two inputs are taken and stored in variables, `inputNum1` and `inputNum2`. The `atan2()` method is then called using the user input, and the result is stored in the variable `result`.

Listing 7.236 Example of the `atan2()` Method

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the arctangent of the
// quotient of the two arguments.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var inputNum2=document.form1.input2.value;

```

Listing 7.236 Continued

```
var result = Math.atan2(inputNum1, inputNum2);
document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Enter First Number:
<input type="text" name="input1" size="10">
<br>
Enter Second Number:
<input type="Text" name="input2" size="10">
<br><br>
<input type="button" value="Calculate" onClick='doMath()'>
<br>
Answer:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.ceil()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.ceil (num)

Description

The `ceil()` method of the `Math` object is used to calculate the smallest integer that is greater than or equal to the number passed in as a parameter. This is similar to getting the ceiling of a number.

Example

Listing 7.237 shows how the `ceil()` method is used to get the smallest integer, which is greater than or equal to the number input by the user.

Listing 7.237 Example of the `ceil()` Method

```
<html>
<body>
<title>Example of the ceil() method</title>
<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the ceiling integer.
```

```
// Takes input and passes it to the ceil method.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.ceil(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates ceiling of the entered number.
<br><br>
Enter First Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The ceil output is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.cos()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.cos (num)

Description

The `cos()` method of the `Math` object is used to calculate the cosine of a number. It returns a numeric value between -1 and 1 , representing the cosine of an angle.

Example

Listing 7.238 shows how the `cos()` method is used to get the cosine of the number input by the user.

Listing 7.238 Example of the `cos()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the cosine of the input number
// takes the input from the user and passes the value to
// the cos() method which calculates the cosine value.
```

Listing 7.238 Continued

```
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.cos(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the cosine of the entered number.
<br><br>
Enter First Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The cosine is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.E

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.E

Description

The E property of the Math object is used to get the value of Euler's constant. This is approximately 2.718.

Example

Listing 7.239 shows how the E property is used.

Listing 7.239 Example of the E Property

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// function to return Euler's constant
function doMath(){
    var result = Math.E;
    document.form1.answer.value = result;
```

```

}
// -->
</script>

<form name="form1">
Click on the button to get Euler's constant.
<br><br>
Euler's constant is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>

```

Math.exp()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE3+, Opera3+

Syntax

math.exp(num)

Description

The `exp()` method of the `Math` object is used to calculate an exponent in which the base is Euler's constant (the base of the natural logarithms).

Example

Listing 7.240 shows how the `exp()` method is used to calculate an exponential value using the number input by the user.

Listing 7.240 Example of the `exp()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function doMath calculates exponential value.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.exp(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Enter First Number:

```

Listing 7.240 Continued

```
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The exponent is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.floor()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.floor (num)

Description

The `floor()` method of the `Math` object is used to get the largest integer number, which is equivalent to or less than the number passed as the parameter.

Example

Listing 7.241 shows how the `floor()` method is used.

Listing 7.241 Example of the `floor()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the floor of the input number.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.floor(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the floor of the number entered.
<br><br>
Enter First Number:
<input type="text" name="input" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
```


The floor is:

```
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.LN10

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.LN10

Description

The `LN10` property of the `Math` object is used to get the natural logarithm of 10. This is approximately equal to 2.302.

Example

Listing 7.242 shows how the `LN10` property is used. When the user clicks the button, the `doMath` function is called, which calculates the natural logarithm of 10 and displays the result in the input box.

Listing 7.242 *Example of the `LN10` Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function gets the natural log of 10.
function doMath(){
    var result = Math.LN10;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The Natural Logarithm of 10 is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'/>
<br>
</form>

</body>
</html>
```

Math.LN2

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.LN2`

Description

The LN2 property of the Math object is used to get the natural logarithm of 2. This is approximately equal to 0.693.

Example

Listing 7.243 shows how the LN2 property is used to get the natural logarithm of 2.

Listing 7.243 *Example of the LN2 Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function gets the natural log of 2.
function doMath(){
    var result = Math.LN2;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The Natural Logarithm of 2 is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>
```

Math.log()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.log(num)`

Description

The `log()` method of the `Math` object is used to calculate the natural logarithm (base *E*) of a number.

Example

Listing 7.244 shows how the `log()` method is used.

Listing 7.244 *Example of the `log()` Method*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Calculates the log of the input and displays the result.
function doMath(){
    var inputNum=document.form1.input.value;
    var result = Math.log(inputNum);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the natural log of the number entered.
<br><br>
Enter First Number:
<input type="text" name="input" size="15">
<input type="button" value="Calculate" onClick='doMath()'/>
<br>
The log is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.LOG10E

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.LOG10E`

Description

The `LOG10E` property of the `Math` object calculates the base 10 logarithm of Euler's constant. The return value is approximately 0.434.

Example

Listing 7.245 shows how the `LOG10E` property is used. When the user chooses the Calculate button, the `doMath` function is executed, which calculates the base 10 logarithm of Euler's constant and outputs the result in the text box.

Listing 7.245 *Example of the `LOG10E` Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function gets the LOG10E value and displays the result.
function doMath(){
    var result = Math.LOG10E;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The Base 10 logarithm of Euler's constant is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath() '>
<br>
</form>

</body>
</html>
```

Math.LOG2E

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.LOG2E`

Description

The `LOG2E` property of the `Math` object calculates the base 2 logarithm of Euler's constant. The return value is approximately 1.442.

Example

Listing 7.246 shows how the LOG2E property is used.

Listing 7.246 Example of the LOG2E Property

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the LOG2E and display the result.
function doMath(){
    var result = Math.LOG2E;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The Base 2 logarithm of Euler's constant is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>
```

Math.max()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
math.max(num1, num2)
```

Description

The max() method of the Math object gets the maximum number of the two parameters passed to it. The larger value is returned as the result.

Example

Listing 7.247 shows how the max() method is used to get the larger of two values.

Listing 7.247 Example of the max() Method

```
<html>
<body>
```

Listing 7.247 Continued

```
<script type="text/javascript" language="JavaScript">
<!--
// Function returns the maximum of two arguments.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var inputNum2=document.form1.input2.value;
    var result = Math.max(inputNum1, inputNum2);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example takes the two numbers entered and determines which is
the larger number.
<br><br>
Enter First Number:
<input type="text" name="input1" size="10">
<br>
Enter Second Number:
<input type="text" name="input2" size="10">
<input type="button" value="Show Max number" onClick='doMath() '>
<br>
The Maximum number is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>
```

Math.min()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.min(num1, num2)

Description

The `min()` method of the `Math` object gets the minimum number of the two number parameters passed to it. The smaller value is returned as the result.

Example

Listing 7.248 shows how to use the `Math.min()` method.

Listing 7.248 Example of the `min()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns the minimum of two arguments.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var inputNum2=document.form1.input2.value;
    var result = Math.min(inputNum1, inputNum2);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example takes the two numbers entered and determind which is
the smaller number.
<br><br>
Enter First Number:
<input type="text" name="input1" size="10">
<br>
Enter Second Number:
<input type="text" name="input2" size="10">
<input type="button" value="Show Min number" onClick='doMath()'>
<br>
The Minimum number is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>

```

Math.PI

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE3+, Opera3+

Syntax

`math.PI`

Description

The `PI` property of the `Math` object is used to get the constant `PI`. This is approximately 3.14159.

Example

Listing 7.249 shows how the `PI` property is used. When the user clicks the Calculate button, the `doMath` function is called, which calculates the value of `PI` and returns the result to the text box.

Listing 7.249 *Example of the `PI` Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns the value of PI.
function doMath(){
    var result = Math.PI;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The Approximate value of PI is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>
```

Math.pow()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.pow(num1, num2)

Description

The `pow()` method of the `Math` object is used to calculate an exponent power.

Example

Listing 7.250 shows how the `pow()` method is used.

Listing 7.250 *Example of the `pow()` Method*

```
<html>
<body>
```



```

<script type="text/javascript" language="JavaScript">
<!--
// Function takes two numbers and calculates the
// exponential power.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var inputNum2=document.form1.input2.value;
    var result = Math.pow(inputNum1, inputNum2);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Enter Base number:
<input type="text" name="input1" size="10">
<br>
Enter exponent to be raised to:
<input type="text" name="input2" size="10">
<input type="button" value="Calculate" onClick='doMath()''>
<br>
The result is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>

```

Math.random()

JavaScript 1.1+, ECMAScript 1E+, JScript 1.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

math.random(num)

Description

The `random()` method of the `Math` object is used to obtain a random number between the values 0 and 1.

Example

Listing 7.251 shows how the `random()` method is used to calculate random numbers.

Listing 7.251 *Example of the random() Method*

```

<html>
<body>

```

Listing 7.251 Continued

```
<script type="text/javascript" language="JavaScript">
<!--
// Function generates a random number between 0 & 1.
function doMath(){
    var result = Math.random();
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The random number is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>
```

Math.round()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.round(num)

Description

The `round()` method of the `Math` object is used to round a number to its nearest integer value. If the fractional portion of the number is `.5` or greater, the result is rounded to the next highest integer. If the fractional portion of number is less than `.5`, the result is rounded to the next lowest integer.

Example

Listing 7.252 shows how the `round()` method is used.

Listing 7.252 Example of the `round()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function rounds input number to nearest integer.
function doMath(){
    var inputNum1=document.form1.input1.value;
```

```

    var result = Math.round(inputNum1);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Enter Number:
<input type="text" name="input1" size="10">
<input type="button" value="Round" onClick='doMath()'>
<br>
The rounded value is:
<input type="text" name="answer" size="10">
</form>

</body>
</html>

```

Math.sin()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.sin (num)

Description

The `sin()` method of the `Math` object is used to calculate the sine of a number. It returns a numeric value between -1 and 1 .

Example

Listing 7.253 shows how the `sin()` method is used.

Listing 7.253 *Example of the `sin()` Method*

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the sine of input number.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var result = Math.sin(inputNum1);
    document.form1.answer.value = result;
}
// -->
</script>

```

Listing 7.253 Continued

```
<form name="form1">
Enter Number:
<input type="text" name="input1" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
The sine is:
<input type="text" name="answer" size="20">
</form>

</body>
</html>
```

Math.sqrt()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.sqrt (num)

Description

The `sqrt()` method of the `Math` object is used to calculate the square root of a number. If the return value is outside the required range, `sqrt()` returns 0.

Example

Listing 7.254 shows how the `sqrt()` method is used.

Listing 7.254 Example of the `sqrt()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function calculates the square root of input number.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var result = Math.sqrt(inputNum1);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Enter Number:
<input type="text" name="input1" size="10">
<input type="button" value="Calculate" onClick='doMath()'>
```

```

<br>
The square root is:
<input type="text" name="answer" size="20">
</form>

</body>
</html>

```

Math.SQRT1_2

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
 Nav2+, NES2+, IE3+, Opera3+

Syntax

```
math.SQRT1_2
```

Description

The SQRT1_2 property of the Math object returns the square root of one half, which is approximately 0.707.

Example

Listing 7.255 shows how SQRT1_2 can be used. The function, doMath, returns the square root of 1/2 to the text box on the page.

Listing 7.255 *Example of the SQRT1_2 Property*

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns the square root of 1/2.
function doMath(){
    var result = Math.SQRT1_2;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The square root of 1/2 is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()''>
<br>
</form>

</body>
</html>

```

Math.SQRT2

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.SQRT2

Description

The SQRT2 property of the Math object returns the value of the square root of 2. This is approximately equal to 1.414.

Example

Listing 7.256 shows how the SQRT2 property is used. The function, doMath, returns the square root of 2 to the text box on the page.

Listing 7.256 *Example of the SQRT2 Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns the square root of 2.
function doMath(){
    var result = Math.SQRT2;
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
The square root of 2 is:
<input type="text" name="answer" size="20">
<input type="button" value="Calculate" onClick='doMath()'>
<br>
</form>

</body>
</html>
```

Math.tan()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

math.tan (num)

Description

The `tan()` method of the `Math` object is to calculate the tangent of a number. It returns a value representing the tangent of an angle.

Example

Listing 7.257 shows how the `tan()` method is used.

Listing 7.257 *Example of the `tan()` Method*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns the tangent of input number.
function doMath(){
    var inputNum1=document.form1.input1.value;
    var result = Math.tan(inputNum1);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
This example calculates the tangent of the entered number.
<br><br>
Enter Number:
<input type="text" name="input1" size="10">
<input type="button" value="Calculate" onClick='doMath()''>
<br>
The tangent is:
<input type="text" name="answer" size="20">
</form>

</body>
</html>
```

Math.toSource()

JavaScript 1.3+
 Nav4.06+

Syntax

```
math.toSource()
```

Description

The `toSource()` method of the `Math` object is used to create a copy of the object. It returns a string representation of an object, which can be passed to the `eval()` method to create a copy of the object.

Example

Listing 7.258 shows how the `toSource()` method is used to make a copy of the `Math` object.

Listing 7.258 *Example of the `toSource()` Method*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function makes copy of math object.
function copy(){
    var result = Math.toSource(Math.E);
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
Click on the button to create a copy of a Math object.
<br><br>
<input type="button" value="Copy" onClick='copy() '>
<br>
The result of toSource is:
<input type="text" name="answer" size="20">
</form>

</body>
</html>
```

Math.toString()

JavaScript 1.0+, JScript 3.0+
Nav2+, NES2+, IE4+

Syntax

`math.toString()`

Description

The `toString()` method of the `Math` object returns a string value representing the object.

Example

Listing 7.259 shows how the `toString()` method is used to get a string value representing the `Math` object.

Listing 7.259 Example of the `toString()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function returns string representation of math object.
function copy(){
    var result = Math.toString(Math.sqrt(45));
    document.form1.answer.value = result;
}
// -->
</script>

<form name="form1">
<input type="button" value="Get String" onClick='copy()'>
<br>
The result of toString is:
<input type="text" name="answer" size="20">
</form>

</body>
</html>

```

Math.unwatch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

math.unwatch (prop)

Description

The `unwatch()` method of the `Math` object is used to turn off the watch on a particular property specified by *prop*. It is inherited from the `Object` object; however it cannot be used because `math` properties are read-only.

Example

All `math` properties are read-only; therefore, no example can be provided.

Math.watch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

math.watch(prop, handler)

Description

The `watch()` method of the `Math` object will watch for an assignment to a specific property, *prop*. When an assignment is made, a specified *handler* will be called to perform a user-defined function. It is inherited from the `Object` object; however, it cannot be used because `math` properties are read-only.

Example

`Math` properties are read-only; therefore, no example is provided.

NaN

JavaScript 1.3+, JScript 1.0+
Nav4.06+, IE3+

Syntax

`NaN`

Description

The `NaN` object represents an object that is not equal to any number, including itself. `NaN` stands for *Not a Number*.

Example

Listing 7.260 shows how the `NaN` object is used within a comparison.

Listing 7.260 Example Using the `NaN` Object

```
<script type="text/javascript" language="JavaScript">
<!--
// Check to see if "a" is a number or not.
if ("a" != NaN){
    document.write("This is not a number");
}
// -->
</script>
```

native

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `native` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented, therefore no example is provided.

netscape

JavaScript 1.1+
Nav3+, NES2+, Opera3+

Syntax

netscape

Description

The netscape object allows you to access any class within the package netscape.*. It is shorter and works the same way as using the packages.netscape property.

Example

Listing 7.261 shows how the netscape property can be used to determine whether the current browser supports the netscape package.

Listing 7.261 Example Using the netscape Object

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Check to see if the netscape package exists.
if (netscape){
    document.write ("This browser has the netscape package");
}
else{
    document.write ("This browser does not have the netscape package");
}

// -->
</script>

</body>
</html>
```

new

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

new

Description

The new operator is used to create a new object.

Example

Listing 7.262 shows how new is used to create a new Array object.

Listing 7.262 *Example of new*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Creates a new array object with the name myArray.
var myArray = new Array();

// -->
</script>

</body>
</html>
```

null

JavaScript 1.2+, JScript 3.0+, ECMAScript 1.0+
Nav4+, NES3+, IE4+

Syntax

Reserved Keyword

Description

The null keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

Number()

JavaScript 1.2+, ECMAScript 1.0+
Nav4+, NES3+, IE5+

Syntax

Number (*obj*)

Description

The `Number()` method takes an object, *obj*, as input and converts it to a number. If the object is a string that is not a well-formed numeric literal, NaN is returned.

Example

Listing 7.263 shows an example of the `Number()` method. A new `Date` object is created and converted to a number.

Listing 7.263 *Example* `Number()` *Method*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
// Create a Date object.
var myStr = new Date();

// Convert the Date to a Number object and display.
document.write("The Number is: " + Number(myStr));

// -->
</script>

</html>
```

Number()

JavaScript 1.1+, ECMAScript 1E+, JScript 1.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

```
var variable = new Number(value)
```

Description

The `Number()` object represents numeric value types. You can create a `Number()` object by specifying a value in the parameter for the number constructor. Table 7.40 shows the different methods and properties of the `Number()` object.

Table 7.40 *Properties and Methods of the* `Number()` *Object*

Type	Item	Description
Property	constructor	Specifies the function that creates the object's prototype.
	MAX_VALUE	Specifies the largest value a number can have.
	MIN_VALUE	Specifies the smallest value a number can have without being equal to 0.
	NaN	Stands for <i>Not a Number</i> . Represents a value that is not equal to any numeric value.

Table 7.40 Continued

Type	Item	Description
Method	NEGATIVE_INFINITY	A special value that represents a negative infinity value.
	POSITIVE_INFINITY	A special value that represents a positive infinity value.
	prototype	Represents the prototype for the number class.
	toExponential()	Returns a string in exponential notation.
	toFixed()	Returns a fixed-length string representation of the number object.
	toLocaleString()	Returns a string representation of the number object in the host's Locale form.
	toPrecision()	Returns a string representation of the number object in an exponential format with a user specified number of digits.
	toSource()	Returns a string representation of the number object.
	toString()	Returns a string representing the specified number object.
	unwatch()	Removes a watchpoint on a specified property.
	valueOf()	Returns the primitive value of a number object as a number data type.
	watch()	Sets a watch point on a specified property.

Example

Listing 7.264 shows how a new Number object is created.

Listing 7.264 Example of the Number Constructor

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Creates a new number object.

var aNum = new Number(3);
// -->
</script>

</body>
</html>
```

Number.constructor

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE4+

Syntax

`Number.constructor`

Description

The constructor property of the Number object specifies the function that creates the object.

Example

Listing 7.265 shows an example of the constructor property.

Listing 7.265 Example Number constructor Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new number object using the constructor property.
num = new Number(3)
if(num.constructor == Number){
    document.write("Object is created");
}
// -->
</script>

</html>
```

Number.MAX_VALUE

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

`Number.MAX_VALUE`

Description

The MAX_VALUE property of the Number object is used to get the maximum representable value for a number. This is approximately: 1.79E+308.

Example

Listing 7.266 shows how the MAX_VALUE property is used.

Listing 7.266 Example of the MAX_VALUE Property

```
<html>
<body>
```

Listing 7.266 Continued

```
<script type="text/javascript" language="JavaScript">
<!--
// Checks to see if the number is a MAX_VALUE.

if((9999*9999) != Number.MAX_VALUE){
    document.write("The number is not greater than the maximum value");
}
// -->
</script>

</body>
</html>
```

Number.MIN_VALUE

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

Number.MIN_VALUE

Description

The MIN_VALUE property of the Number object is used to get the minimum possible numeric value known to JavaScript. This is approximately: 2.22E–308.

Example

Listing 7.267 shows how the MIN_VALUE property is used.

Listing 7.267 Example of MIN_VALUE

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Checks to see if the number is equal to the MIN_VALUE.

if((0.00000002) != Number.MIN_VALUE){
    document.write("The number is not the minimum value");
}
// -->
</script>

</body>
</html>
```


Number.NaN

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

`Number.NaN`

Description

The NaN property of the Number object represents a value that is not equal to any numeric value.

Example

Listing 7.268 shows how to use the NaN property. An integer constant, 123, is compared to the NaN constant to see whether it is a numeric value.

Listing 7.268 Example of the NaN Property

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Checks to see if 123 is a number or not.

if(123 == Number.NaN){
    document.write("This is not a number");
}
// -->
</script>

</body>
</html>
```

Number.NEGATIVE_INFINITY

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

`Number.NEGATIVE_INFINITY`

Description

The NEGATIVE_INFINITY property of the Number object represents a negative infinity number. It is returned when a calculation returns a negative number greater than the largest negative number in JavaScript.

Example

Listing 7.269 shows how the `NEGATIVE_INFINITY` property is used. The `sqrt()` method is used on a number and the result is compared to `NEGATIVE_INFINITY`.

Listing 7.269 *Example of* `NEGATIVE_INFINITY`

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Performs a square root calculation to obtain a negative result
// and then checks against the NEGATIVE_INFINITY value.

if((Math.sqrt(-2)) != Number.NEGATIVE_INFINITY){
    document.write("This is not equal to NEGATIVE_INFINITY");
}
else{
    document.write("This is equal to NEGATIVE_INFINITY");
}

// -->
</script>

</body>
</html>
```

Number.POSITIVE_INFINITY

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

`Number.POSITIVE_INFINITY`

Description

The `POSITIVE_INFINITY` property of the `Number` object represents a positive infinity number. It is returned when a calculation returns a positive number greater than the largest number in JavaScript.

Example

Listing 7.270 shows how the `POSITIVE_INFINITY` property is used.

Listing 7.270 *Example of* `POSITIVE_INFINITY`

```
<html>
<body>
```

```

<script type="text/javascript" language="JavaScript">
<!--
// Performs some math computation and then checks the
// result against the POSITIVE_INFINITY value.
if((Math.exp(999)) != Number.POSITIVE_INFINITY){
    document.write("This is less than POSITIVE_INFINITY");
}
else{
    document.write('This is greater than POSITIVE_INFINITY');
}

// -->
</script>

</body>
</html>

```

Number.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
 Nav3+, NES2+, IE3+, Opera3+

Syntax

`Number.prototype.property`

`Number.prototype.method`

Description

The prototype property of the Number object allows you to add properties or methods to all instances of this class.

Example

Listing 7.271 shows how the prototype property is used.

Listing 7.271 Example of prototype

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Creates a new Number property myProp.
var myProp = new Number();

// Sample function multiplies number by 3.
function triple(num){

```

Listing 7.271 Continued

```
    var result;
    result = (num * 3);
    return result;
}

// Add the prototype property to the number object.
Number.prototype.calc3 = triple;

document.write("Example demonstrates the prototype property for the number
object." + "<br><br>");
document.write("150 tripled is: " + myProp.calc3(150) + "<br>");
// -->
</script>

</body>
</html>
```

Number.toExponential()

JavaScript 1.3+, ECMAScript 3E+
Nav6+, IE5.5+

Syntax

number.toExponential()

Description

The toExponential() method of the Number object is used to get an exponential representation of the Number object.

Example

Listing 7.272 shows how the toExponential() method is used.

Listing 7.272 Example of the toExponential() Method

```
<script type="text/javascript" language="JavaScript">
<!--

var abc = 51000;
// Convert the number to Exponential form and display.
document.write("The variable in exponential form is: " + abc.toExponential() );

// -->
</script>
```

Number.toFixed()

JavaScript 1.4+, ECMAScript 3E+
Nav6+, IE5.5+

Syntax

`number.toFixed(num)`

Description

The `toFixed()` method of the `Number` object is used to get a fixed-point string representation of the `Number` object. The *num* parameter represents the number of digits after the decimal point.

Example

Listing 7.273 shows how the `toFixed()` method is used to get the fixed form of the standard `Number` object.

Listing 7.273 Example of the `toFixed()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

var standard = 888000.4325;
// Convert the number to a fixed form and display.
document.write("Fixed Form is: " + standard.toFixed(0) );

// -->
</script>
```

Number.toLocaleString()

JavaScript 1.4+, ECMAScript 3E+, JScript 1.0+
Nav6+, IE5.5+, Opera5+

Syntax

`number.toLocaleString()`

Description

The `toLocaleString()` method of the `Number` object is used to get a string value that represents the `Number` object. The value will be formatted according to the conventions of the host environment's current locale.

Example

Listing 7.274 shows how the `toLocaleString()` method is used to display the string value of `myNum`.

Listing 7.274 Example of the `toLocaleString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

var myNum = 123456789;
// Get the string value and display.
document.write("The Locale String is: " + myNum.toLocaleString() );

// -->
</script>
```

Number.toPrecision()

JavaScript 1.4+, ECMAScript 3E+
Nav6+, IE5.5+

Syntax

`number.toPrecision(num)`

Description

The `toPrecision()` method of the `Number` object returns a string representation of the number in the format of one digit before the significand's decimal point and *num-1* digits after the digit's decimal point.

Example

Listing 7.275 shows how the `toPrecision()` method is used.

Listing 7.275 Example of the `toPrecision()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

var myNum = 123456789.4444;
// Return the result of the toPrecision() method on myNum.
document.write("The Precision String is: " + myNum.toPrecision(2) );

// -->
</script>
```

Number.toSource()

JavaScript 1.3+, ECMAScript 1E+
Nav4.06+

Syntax

`number.toSource()`

Description

The `toSource()` method of the `Number` object is used to get a string representation of the `Number` object.

Example

Listing 7.276 shows how the `toSource()` method is used.

Listing 7.276 Example of the `toSource()` Method

```
<script type="text/javascript" language="JavaScript">
<!--
// Creates a new number object and then gets the string
// representation of that object.
var aNum = Number(21);
document.write(aNum.toSource());
// -->
</script>
```

Number.toString()

JavaScript 1.1+, ECMAScript 1E+, JScript 1.0+
 Nav3+, NES2+, IE3+, Opera3+

Syntax

`number.toString()`

Description

The `toString()` method of the `Number` object is used to get a string representation of the `Number` object.

Example

Listing 7.277 shows how the `toString()` method is used.

Listing 7.277 Example of the `toString()` Method

```
<script type="text/javascript" language="JavaScript">
<!--
var aNum = Number(21);
// Return the string representation of aNum.
document.write("The string value for 21 is: " + "<b>"
+ aNum.toString() + "</b>");
// -->
</script>
```

Number.unwatch()

JavaScript 1.1+, JScript 3.0+
 Nav3+, NES2+, IE4+

Syntax

`number.unwatch(prop)`

Description

The `unwatch()` method of the `Number` object will remove a watch point on a property set by the `watch()` method.

Example

Listing 7.278 shows an example for the `unwatch()` method.

Listing 7.278 *Example of* `unwatch()` *Method*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.writeln("myNum." + id + " changed from " + oldValue
➡+ " to " + newValue + "<br>")
    return newValue;
}

// Create a new Number object.
var myNum = new Number(10);

// Create a new property p.
Number.prototype.p = 1;
myNum.watch("p",alertme);

myNum.p = 2;
// Remove the watchpoint.
myNum.unwatch("p");
myNum.p = 3;
myNum.p = 4;

// Set the watchpoint.
myNum.watch("p",alertme);
myNum.p = 5;

// -->
</script>

</html>
```

Number.valueOf()

JavaScript 1.1+, JScript 2.0+
Nav3+, NES2+, IE4+

Syntax

```
number.valueOf()
```

Description

The `valueOf()` method of the `Number` object is used to get the primitive value of a `Number` object as a number data type.

Example

Listing 7.279 shows an example for the `valueOf()` method. A `Number` object is created and set to `myNum`. The document then outputs the result of performing a `valueOf` function on the number.

Listing 7.279 Example of the `Number.valueOf()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new number object.
myNum = new Number(24)

// Output the valueOf result.
document.write("The value of myNum is: " + myNum.valueOf());

// -->
</script>

</body>
</html>
```

Number.watch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

```
number.watch(prop, handler)
```

Description

The `watch()` method of the `Number` object will watch for an assignment to a specific property, *prop*. When an assignment is made, a specified *handler* will be called to perform a user-defined operation.

Example

Listing 7.280 shows an example for the `watch()` method.

Listing 7.280 Example of the `watch()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function to alert user that p has changed.
function alertme(id,oldValue,newValue)
{
    document.writeln("myNum." + id + " changed from " + oldValue
➡+ " to " + newValue + "<br>")
    return newValue;
}

var myNum= new Number(10);
// Create a new property p.
Number.prototype.p = 1;

// Set the watchpoint on p.
myNum.watch("p",alertme);

myNum.p = 15;
// -->
</script>

</body>
</html>
```

Object()

JavaScript 1.1+, ECMAScript 1E+, JScript1.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

```
var variable = new Object(string)
```

Description

The `Object()` object is a primitive data type from which all JavaScript objects are derived. Table 7.41 shows the different properties and methods of the `Object()` object.

Table 7.41 Properties/Methods of the `Object()` Object

Type	Item	Description
Property	constructor	Creates an Object
	prototype	Creates a new property for a specific object
Method	eval()	Evaluates a string of JavaScript code for the specified object

Type	Item	Description
	<code>toSource()</code>	Returns a string representation for the object
	<code>toString()</code>	Converts the object to its string representation
	<code>unwatch()</code>	Removes a watchpoint for the object
	<code>valueOf()</code>	Returns the value of the specific object
	<code>watch()</code>	Adds a watchpoint to the object property

Example

Listing 7.281 shows how the `Object` object is used.

Listing 7.281 *Example of the `Object` Object*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Create an Object.
var myObj = new Object(foo);
document.write(Object foo created);
// -->
</script>

</body>
</html>
```

Object.constructor

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
 Nav3+, NES2+, IE4+

Syntax

`object.constructor`

Description

The `constructor` property of the `Object` object specifies the function that creates the object.

Example

Listing 7.282 shows an example of the `constructor` property.

Listing 7.282 *Example `Object` constructor Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
```

Listing 7.282 Continued

```
// Create a new number object using the constructor property.
num = new Number(3)
if(num.constructor == Number){
    document.write("Object is created");
}
// -->
</script>

</body>
</html>
```

Object.eval()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES2+, IE4+

Syntax

object.eval (*string*)

Description

The eval() method of the Object object evaluates a *string* of JavaScript code in reference to this object. Note: In version 1.4 and later, the eval() method is deprecated and can no longer be called directly for the Object object. To use eval() in versions 1.4 and 1.5, you must use the top-level eval() function.

Example

Listing 7.283 shows how the eval() method is used. Two variables are declared and set. A statement multiplying the two variables together is passed to the eval() method to be evaluated.

Listing 7.283 Example of the eval() Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

var x = 9;
var y = 8;

// Display the result of x*y.
document.write("The result of x * y is: " + eval(x * y));
// -->
</script>

</body>
</html>
```

Object.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE4+

Syntax

object.prototype.property

object.prototype.method

Description

The prototype property of the Object object allows the addition of properties or methods to the Object class.

Example

Listing 7.284 shows how the prototype property is used.

Listing 7.284 *Example of the prototype Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
Object.prototype.newProperty = 2;
// Display the value of the newProperty.
document.write(document.object.newProperty.value);
// -->
</script>

</body>
</html>
```

Object.toSource()

JavaScript 1.3+
Nav4.06+

Syntax

object.toSource()

Description

The toSource() method is used to get a string representation of the object.

Example

Listing 7.285 shows how the toSource() is used.

Listing 7.285 Example of the `toSource()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Creates a new number object and then gets the string
// representation of that object.
var aNum = Number(21);
document.write(aNum.toSource());
// -->
</script>

</body>
</html>
```

Object.toString()

JavaScript 1.1+, ECMAScript 1E+, JScript 2.0+
Nav3+, NES2+, IE4+, Opera3+

Syntax

```
object.toString()
```

Description

The `toString()` method is used to get a string representation of the `Number` object.

Example

Listing 7.286 shows how the `toString()` method is used.

Listing 7.286 Example of the `toString()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Creates a number object.
var aNum = Number(21);

// Converts the number object to a string and outputs to document.
document.write(aNum.toString());
// -->
</script>

</body>
</html>
```

Object.unwatch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

`object.unwatch (prop)`

Description

The `unwatch()` method of the `Object` object allows you to remove a watchpoint set on a property with the `watch()` method. This method takes the property, *prop*, as a parameter.

Example

Listing 7.287 shows how the `unwatch()` object is used. A temporary variable, `tmp`, is created and initialized. It is then set to be watched by invoking the `watch()` method. If any changes occur to the `tmp` variable, the `inform` function is called. After a change is made to the variable, `unwatch()` is called to turn off watch operations on the variable. After watch operations are disabled, the variable can be changed without notification.

Listing 7.287 Example of the `unwatch()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function informs the user when the tmp variable is changed.
function inform(){
    document.write("Tmp variable changed from 1 to 3");
}

// Declare a tmp variable.
var tmp = 1;

// Watch the tmp variable for any changes.
watch("tmp",inform);
tmp=3;

// Turn off watch on the tmp variable.
unwatch("tmp");
tmp=7;
// -->
</script>

</body>
</html>
```

Object.valueOf()

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
Nav3+, NES2+, IE4+, Opera3+

Syntax

`object.valueOf()`

Description

The `valueOf()` method for the `Object` object is used to obtain the value of the specified object.

Example

Listing 7.288 shows how the `valueOf()` method is used.

Listing 7.288 Example of the `valueOf()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Declare an age variable which contains a Number object.
var age = Number(30);

// Calculate the valueOf the variable and output to the document.
document.write(age.valueOf());
// -->
</script>

</body>
</html>
```

Object.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`object.watch(prop, function)`

Description

The `watch()` method of the `Object` object is used to watch for the event in which a property gets assigned a value. When the assignment is made, a user-defined function is executed. The method itself takes the property to watch, *prop*, and the function to call, *func*, when the event occurs.

Example

Listing 7.289 shows how the `watch()` method is used.

Listing 7.289 *Example of the* `watch()` *Method*

```
<html>
<body>
<title>Example of the watch method</title>
<script type="text/javascript" language="JavaScript">
<!--

// Function informs the user when the tmp variable is changed.
function inform(){
    document.write("Tmp variable changed from 1 to 3");
}

// Declare a tmp variable and initialize.
var tmp = 1;

// Turn on watch operations on the variable. If the tmp
// variable is changed, then the inform function is run.
watch("tmp",inform);

// Change the tmp variable.
tmp=3;
// -->
</script>

</body>
</html>
```

package

JavaScript 1.2+
Nav4+, NE53+

Syntax

Reserved Keyword

Description

The package keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

Packages

JavaScript 1.1+
Nav3+, NES2+

Syntax

`Packages.packageName`

Description

The Packages object is a built-in object that provides access to various Java packages within the browser. Each property of the Packages object refers to a `JavaPackage` object containing references to specific classes. Table 7.42 shows the default packages included in the Packages object.

Table 7.42 *Properties of the Packages Object*

Package	Description
<code>classname</code>	Refers to fully qualified name of Java class in package
<code>java</code>	Refers to the core Java classes
<code>netscape</code>	Refers to a set of Netscape classes
<code>sun</code>	Refers to the core Sun classes

Example

Listing 7.290 shows how the Packages object is used. The user is provided with an input text box. When something is entered in the input box and the button clicked, the input is sent to the Java Console using the Java classes package.

Listing 7.290 *Example of the Packages Object*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function takes the users input and writes it out to
// the Java Console
function writeOut(input){
    Packages.java.lang.System.out.println(input);
}
// -->
</script>
```

```
<form name="form1">
```

This script takes the text input and writes it out to the Java Console using the Java package.

```
<br><br>
Input:
```

```
<input type="text" size="40" name="txt">
<br><br>
<input type="button" value="Write Out Text" name="button1"
  ⚡ onClick='writeOut(document.form1.txt.value)'\>
<br>
</form>

</body>
</html>
```

Packages.className

JavaScript 1.1+
Nav3+, NES2+

Syntax

`Packages.className`

Description

The `className` property of the `Packages` object is used to access classes in packages other than `netscape`, `sun`, or `java`. Just specify the fully qualified name of the package you want to access for `className`.

Example

Listing 7.291 shows an example for accessing a fictitious class using the `className` property.

Listing 7.291 Example of `Packages.className`

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Access the class: myClass.
myName = new Packages.myClass.Car()

// -->
</script>

</body>
</html>
```

Packages.java

JavaScript 1.1+
Nav3+, NES2+

Syntax

`Packages.java.className.methodName`

Description

The `java` sub-package of the `Packages` object refers to the `JavaPackage` containing the core Java class library. This sub-package is used for several things, but most notably for adding security to `LiveConnect` and accessing the Java Console.

Example

Listing 7.292 shows an example for the `java` sub-package. It is used to write text to the Java Console.

Listing 7.292 *Example of the* `Packages.java` *Sub-package*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Use the Java sub-package to write text to the Java Console.
Packages.java.lang.System.out.println("Hello World!");

// -->
</script>

</body>
</html>
```

Packages.netscape

JavaScript 1.1+
Nav3+, NES2+

Syntax

`Packages.netscape.className.methodName`

Description

The `netscape` sub-package of the `Packages` object refers to the `JavaPackage` containing the `netscape` package. This sub-package is used by Java applets to access JavaScript code via `LiveConnect`. The package itself has two classes: `plugin` and `javascript`.

Example

Use of this package occurs within the code of a Java applet, and not JavaScript code. However, Listing 7.293 shows an example of calling the `netscape` package directly to verify it is implemented in the operating browser.

Listing 7.293 Example of Accessing the netscape Package

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Call the package to see if it exists.
if(Packages.netscape){
    document.write("This browser has LiveConnect!");
}else{
    document.write("This browser does not have LiveConnect!");
}
// -->
</script>

</body>
</html>

```

Packages.sun

JavaScript 1.1+
Nav3+, NES2+

Syntax

`Packages.sun.className.methodName`

Description

The sun sub-package of the Packages object refers to the JavaPackage for the sun property. This sub-package is used for several things, but most notably for adding security to LiveConnect.

Example

Use of this package occurs within the code of a Java applet, and not JavaScript code. However, Listing 7.294 shows an example of calling the sun package directly to verify it is implemented in the operating browser.

Listing 7.294 Example of Accessing the sun Package

```

<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Call the package to see if it exists.
if(Packages.sun){
    document.write("This browser has LiveConnect!");
}else{
    document.write("This browser does not have LiveConnect!");
}

```

Listing 7.294 Continued

```
}  
// -->  
</script>  
  
</body>  
</html>
```

parseFloat()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`parseFloat (string)`

Description

The `parseFloat()` function is used to convert a string to a number.

Example

Listing 7.295 shows how the `parseFloat()` is used. In the example, `parseFloat()` is called with two different strings. The first string, which contains numeric characters, is converted into a number without any problem. The second string, which contains alphabetic characters, is unable to be converted into a number.

Listing 7.295 Example of the `parseFloat()` Method

```
<html>  
<body>  
  
<script type="text/javascript" language="JavaScript">  
<!--  
// Convert the "1245.31" string to a number.  
document.write("The string 1245.31 converted is"  
➡+ parseFloat("1245.31") + "<br>");  
  
// Try to convert the string "test" to a number.  
// If not possible, then print error.  
if( isNaN(parseFloat("test")) ){  
    document.write("Cannot convert test string to a number.");  
}  
// -->  
</script>  
  
</body>  
</html>
```

parseInt()

JavaScript 1.0+, ECMAScript 1E+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
parseInt(string, radix)
```

```
parseInt(string)
```

Description

The `parseInt()` method is used to convert a string to an integer. It can take *string* input with an optional *radix* input. The *radix* input represents the base of the number in the string.

Example

Listing 7.296 shows how `parseInt()` is used to parse a string. A few different examples are shown for different types of strings.

Listing 7.296 *Example of the* `parseInt()` *Method*

```
<html>
<body>
<script type="text/javascript" language="JavaScript">
<!--

// Converts the "859" string to an integer.
document.write("The string 859 converted to an integer is: ");
document.write(parseInt("859") + "<br>");

// Converts a binary string into an integer.
document.write("The binary string 101101 converted to an integer is: ");
document.write(parseInt("101101", 2) + "<br>");

// Converts a hexadecimal string into an integer.
document.write("The hexadecimal string FA832B converted to an integer is: ");
document.write(parseInt("FA832B", 16) + "<br>");

// -->
</script>

</body>
</html>
```

private

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `private` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

protected

JavaScript 1.2
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `protected` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

public

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `public` keyword has not been implemented in server-side JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

RegExp()

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

```
var variable = new RegExp(pattern, flags)
```


Description

The `RegExp()` object represents a regular expression that is used for pattern matching. The creation of the object takes *pattern* and *flags* parameters. The *pattern* is a valid regular expression. The *flags* are either or both `g` (global) and `i` (ignore case). Table 7.43 displays the properties and methods of the `RegExp()` object.

Table 7.43 *Properties and Methods of the `RegExp()` Object*

Type	Item	Description
Property	<code>RegExp.\$*</code>	Represents multiline
	<code>RegExp.\$&</code>	Represents lastmatch
	<code>RegExp.\$_</code>	Represents input
	<code>RegExp.\$`</code>	Represents leftContext
	<code>RegExp.\$'</code>	Represents rightContext
	<code>RegExp.\$+</code>	Represents lastParen
	<code>RegExp.\$1,\$2,...\$9</code>	Represents substring of matches
	constructor	Creates the object
	global	Specifies whether to check the expressions against all possible matches
	ignoreCase	Whether case is ignored during a string search
	input	String that is matched
	lastIndex	Specifies the index at which to start matching the next string
	lastMatch	Last matched characters
	lastParen	The last parenthesized substring match
	leftContext	The substring preceding the most recent match
	multiline	Specifies whether to search on multiple lines
	prototype	Represents the prototype for the <code>RegExp</code> object
	rightContext	The substring following the most recent match
	source	The string pattern
Method	<code>compile()</code>	Compiles a regular expression
	<code>exec()</code>	Executes the search for a match in a specified string
	<code>test()</code>	Tests for a string match
	<code>toString()</code>	Returns a string representation of the object
	<code>unwatch()</code>	Removes a watch point on the object
	<code>valueOf()</code>	Returns the primitive value of the object
	<code>watch()</code>	Sets a watchpoint on the object

Example

Listing 7.297 shows how to use the RegExp object. The user is given an input field, which is used to input a Social Security Number (SSN). Once entered, the Validate button is clicked, which checks whether the input is valid. This is performed by using a RegExp object for the SSN.

Listing 7.297 *Example of the* `RegExp` *Object*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function checks to see if the ssn is valid.
function isSSN(str){

    // Define a RegExp object which checks for either
    // a 9 digit input or an input in the form:
    // xxx-xx-xxxx
    var regexp = /^(\d{9}|\d{3}-\d{2}-\d{4})$/;
    return regexp.test(str);
}

// Checks the SSN input.
function checkInput(){
    var valid = true;
    var ssn = document.form1.ssn.value;
    if (!isSSN (ssn)){
        window.alert("Invalid SSN: " + ssn);
        valid = false;
    }
    else{
        alert(ssn + " is a valid SSN");
    }
}

// -->
</script>

<form name="form1">
Enter your SSN:
<input type="text" size="15" name="ssn">
<br><br>
<input type="button" value="Validate SSN" onClick='checkInput() '>
<br>
</form>
</body>
</html>
```

RegExp,\$*

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

RegExp, \$*

Description

The `RegExp,$*` property reflects a multiline string search. This is a Boolean, read-only value that reflects whether strings should be searched across multiple lines. This is the same as using the `multiline` property.

Example

Listing 7.298 shows how to use `RegExp,$*` for pattern matching.

Listing 7.298 *Example of* `RegExp,$*`

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
// Function checks for the "the" expression. However, if
// multiple lines are read, an alert box is displayed
// indicating so.
function getinfo(){

var myPat = new RegExp("the", "i");
var str = document.form1.mytext.value;
myArray = myPat.exec(str);

    alert("RegExp.$* is: " + RegExp["$*"]);
}
// -->
</script>

<body>
<form name="form1">
When the text in the text box is changed, and the document is clicked,
an alert box will be displayed showing the value of RegExp.$*.
<br><br>
<textarea name="mytext" cols="60" rows="8" onChange='getinfo() '>
This is a sample textarea containing some dummy text for
testing purposes. The text in this box will be used to
demonstrate how the multiline property is used. If multiple lines are
read, then RegExp.$* will be true.
</textarea>
```

Listing 7.298 Continued

```
<br>
</form>

</body>
</html>
```

RegExp.\$&

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

RegExp.\$&

Description

The RegExp.\$& property represents the last matched characters. This is the same as using the lastMatch property.

Example

Listing 7.299 shows how RegExp.\$& is used.

Listing 7.299 Example of RegExp.\$&

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a pattern to search for.
var pat = new RegExp("test", "gi");
str = "Testing Testing 123";
myArray = pat.exec(str);

// Once pattern is found, display message.
document.write("Pattern found: " + myArray[0] +
               ". the last match expression ($&) is: " + RegExp["$&"]);
// -->
</script>

</body>
</html>
```

RegExp,\$_

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

RegExp, \$ _

Description

The RegExp, \$ _ property represents the input to which a string is matched. This is the same as using the input property.

Example

Listing 7.300 shows how to use the RegExp, \$ _ property.

Listing 7.300 *Example of* RegExp, \$ _

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function creates a new regular expression and
// then executes it against the text in the textbox.
// Outputs an alert message indicating the value
// of the RegExp.input property.
function getinput(){

    var myPat = new RegExp("the", "i");
    var str = document.form1.mytext.value;
    myArray = myPat.exec(str);

    alert("The RegExp$ is: " + RegExp["$_"]);
}
// -->
</script>

<form name="form1">
When the text in the text box below is changed, an alert message
will appear showing the value of the input.
<br><br>
Enter some Text:
<input type="text" name="mytext" size="40" onChange='getinput() '>
<br>
</form>

</body>
</html>
```

RegExp.\$`

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

RegExp.\$`

Description

The `RegExp.$`` property represents the substring preceding the most recent pattern match. This is the same as using the `leftContext` property.

Example

Listing 7.301 shows how to use `RegExp.$``.

Listing 7.301 *Example of* `RegExp.$``

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a regular expression pattern and match globally.
pat = /is*/g;

// Create a string object.
var str = "I know where the fish is tonight.";

// Create an array to hold the results.
myArray = pat.exec(str);

document.write("Check for the substring " + "<i>" + "is" + "</i>");
document.write("preceding most recent pattern match");
document.write("In the string: " + "<b>");
document.write("I know where the fish is tonight" + "</b><br><br>");
document.write("The RegExp. $` is: " + RegExp["$`"]);

// -->
</script>

</body>
</html>
```

RegExp.\$'

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

RegExp.\$'

Description

The `RegExp.$'` property represents the substring following the most recent pattern match. This is the same as using the `rightContext` property.

Example

Listing 7.302 shows how to use `RegExp.$'`.

Listing 7.302 *Example of* `RegExp.$'`

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a regular expression pattern and match globally.
pat = /be*/gi;

// Create a string object.
var str = "Eat Drink and be Merry.";

// Create an array to hold the results.
myArray = pat.exec(str);

document.write("Check for the substring " + "<i>" + "be" + "</i>");
document.write("following the most recent pattern match");
document.write("In the string: " + "<b>" +
    "Eat Drink and be Merry" + "</b><br><br>");
document.write("The RegExp$' is: " + RegExp["$'"]);

// -->
</script>

</body>
</html>
```

RegExp.\$+

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

`RegExp.$+`

Description

The `RegExp.$+` property represents the last parenthesized substring pattern match. This is the same as using the `lastParen` property.

Example

Listing 7.303 shows how `RegExp.$+` is used.

Listing 7.303 *Example of* `RegExp.$+`

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a regular expression.
exp = new RegExp("(please)", "g");

// Create a string object.
str = "Will you (please) stop yelling!";
myArray = exp.exec(str);

// Inform user what the lastParen property is.
document.write("The RegExp.$+is: " + "<b>"
    + RegExp["$+"] + "</b>");

// -->
</script>

</body>
</html>
```

RegExp.\$1,\$2,..\$9

JavaScript 1.2+, JScript 1.0+
Nav4+, NES3+, IE4+

Syntax

`RegExp.$1,$2,..$9`

Description

The `RegExp.$1,$2,..$9` property represents parenthesized substring matches.

Example

Listing 7.304 shows how `RegExp.$1,$2,..$9` is used. The user will enter his phone number in the input text box and, when the button is clicked, the swap function swaps the last four digits in the phone number with the first three.

Listing 7.304 *Example of* `RegExp.$1,$2,..$9`

```
<html>

<script type="text/javascript" language="JavaScript">
```



```

<!--
// Function takes the input and swaps the last 4 digits with the
// first three digits.
function swap(){
re = /(\w+)\D(\w+)/;
str = document.form1.text1.value;
newstr=str.replace(re, "$2, $1");
document.form1.text2.value = newstr;
}
// -->
</script>

<body>
<form name="form1">
Enter your 7 digit phone number in the form xxx-xxxx
<br><br><br>
Phone Number (7 digits):<input type="text" name="text1" size="10">
<br><br>
<input type="button" value="Swap" onClick='swap()''>
<br><br><br>
Output: <input type="text" name="text2" size="10">
</form>

</body>
</html>

```

RegExp.constructor()

JavaScript 1.1+, JScript 3.0+
Nav3+, NES2+, IE4+, Opera5+

Syntax

regexp.constructor

Description

The constructor property of the RegExp object specifies the function that creates the object.

Example

Listing 7.305 shows how the constructor property is used to create a RegExp object.

Listing 7.305 Example of the constructor Property

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

```

Listing 7.305 Continued

```
// Create a new RegExp object using the constructor property.
myExp = new RegExp("the")
if(myExp.constructor == RegExp){
    document.write("Object is created");
}
// -->
</script>

</body>
</html>
```

RegExp.compile()

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

```
regexp.compile(pattern, flag)
```

Description

The `compile()` method of the `RegExp` object compiles a regular expression object. The creation of the object takes *pattern* and *flags* parameters. The *pattern* is a valid regular expression. The *flags* are either or both *g* (global) and *i* (ignore case).

Example

Listing 7.306 shows how to use the `compile()` method. A pattern is created using the `RegExp` constructor. It is then compiled using the `compile()` method, and the result is displayed in the text area.

Listing 7.306 Example of the `compile()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
var myPat = new RegExp("jane", "i");
var newPat = myPat.compile(myPat);

// function displays the result of the compiled pattern.
function getinfo(){
    document.form1.text1.value = newPat;
}
// -->
</script>
<body>
```

```

<form name="form1">
Click the button below to get the pattern for the following
command: new RegExp("jane", "i");
<br><br><br>
Compiled Pattern: <input type="text" name="text1" size="30">
<br><br>
<input type="button" value="Get Pattern" onClick='getinfo()'>
</form>

</body>
</html>

```

RegExp.exec()

JavaScript 1.2+
 Nav4+, NES3+, IE4+

Syntax

regexp.exec (string)

Description

The `exec()` method of the `RegExp` object executes the search for a match in a specified string. The results are returned in an array. The *string* passed contains the string the regular expression is trying to match in.

Example

In Listing 7.307, you see how the `exec()` method is used. A regular expression is defined and executed on the string using the `exec()` method.

Listing 7.307 Example of the `exec()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Checks for the pattern "xyz" in str. If found, then
// output written to document indicating that it was found and
// displays the index it was found in the string.

myRe=/xyz*/g;
str = "abcxyzdefhij"
myArray = myRe.exec(str);

document.writeln("Found " + myArray[0] + " in the pattern: " + "<b>" +
"abcxyzdefhij " + "</b>" + " at index " +
(myRe.lastIndex - 3));

```

Listing 7.307 Continued

```
// -->
</script>
</body>
</html>
```

RegExp.global

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

`regexp.global`

Description

The `global` property of the `RegExp` object specifies whether the `g` flag is used with the regular expression. If so, a global pattern match will be performed.

Example

Listing 7.308 shows how the `global` property is used. A new `RegExp` object is created specifying the `global` option. When the script is loaded, the expression is checked and the value of the `global` property is printed out.

Listing 7.308 Example of the `global` Property

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Defines a regular expression on the pattern "if"
// with the global flag set.
var myPat = new RegExp("if", "g");

// Define a string.
var str = "What if Angela is wondering about gifts?";

// Store results of exec into myArray.
myArray = myPat.exec(str);

document.write("The value of RegExp.global is: " + "<b>"
    + myPat.global + "</b>");
// -->
</script>

</body>
</html>
```

RegExp.ignoreCase

JavaScript 1.2+
 Nav4+, NES3+, IE4+

Syntax

`regex.ignoreCase`

Description

The `ignoreCase` property of the `RegExp` object is a flag that informs the user whether case is to be ignored during pattern matching.

Example

Listing 7.309 shows how `ignoreCase` is used. A new `RegExp` object is created specifying the `ignoreCase` option. When the script is loaded, the expression is checked and according to the value of the `ignoreCase` property, an appropriate message is displayed.

Listing 7.309 Example of `ignoreCase`

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Defines a regular expression on the pattern "and"
// with the ignore case flag set.
var myPat = new RegExp("and", "i");

// Define a string.
var str = "Would Missy and Livvy like some Candy?";

// Store results of exec into myArray.
myArray = myPat.exec(str);

if (myPat.ignoreCase == true){
    document.write("The " + "<i>" + "ignoreCase" + "</i>" + " option WAS
used");
}
else{
    document.write("The " + "<i>" + "ignoreCase" + "</i>" + " was NOT used");
}

// -->
</script>

</body>
</html>
```

RegExp.input

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

`regex.input`

Description

The input property of the RegExp object represents the string on which the pattern matching is performed.

Example

Listing 7.310 shows how to use the input property.

Listing 7.310 *Example of the* input *Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function creates a new regular expression and
// then executes it against the text in the textbox.
// Outputs an alert message indicating the value
// of the RegExp.input property.
function getinput(){

var myPat = new RegExp("the", "i");
var str = document.form1.mytext.value;
myArray = myPat.exec(str);

    alert("The RegExp.input is: " + RegExp.input);
}

// -->
</script>

<form name="form1">
When the text in the text box below is changed, an alert message
will appear showing the value of the input.
<br><br>
Enter some Text:
<input type="text" name="mytext" size="40" onChange='getinput() '>
<br>
</form>

</body>
</html>
```

RegExp.lastIndex

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

regex.lastIndex

Description

The `lastIndex` property of the `RegExp` object is used to get the index of where the next match begins.

Example

Listing 7.311 shows how the `lastIndex` property is used. A regular expression for "is" is created and checked against the string. When found, results are written to the document.

Listing 7.311 Example of `lastIndex`

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
</script>

// Creates a regular expression for "is".
exp=/is*/g;
str = "This is just a sample sentence.";
myArray = exp.exec(str);

document.write("Found: " + myArray[0] +
               ". Next match starts at index: " + exp.lastIndex);
// -->
</script>
</body>
</html>
```

RegExp.lastMatch

JavaScript 1.2+, JScript 5.5+
Nav4+, NES3+, IE4+

Syntax

regex.lastMatch

Description

The `lastMatch` property of the `RegExp` object represents the last matched characters.

Example

Listing 7.312 shows how the `lastMatch` property is used. A `RegExp` object is created to look for the string "test" within the `str` variable. Upon loading the document, the `lastMatch` property will display the last pattern to be matched.

Listing 7.312 *Example of the `lastMatch` Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a pattern to search for.
var pat = new RegExp("test", "gi");
str = "Testing Testing 123";
myArray = pat.exec(str);

// Once pattern is found, display message.
document.write("Pattern found: " + myArray[0] +
               ". the last match expression is: " + RegExp.lastMatch);
// -->
</script>

</body>
</html>
```

RegExp.lastParen

JavaScript 1.2+, JScript 5.5+
Nav4+, NES3+, IE4+

Syntax

regex.lastParen

Description

The `lastParen` property of the `RegExp` object represents the last parenthesized substring match. It returns a string value for the last parenthesized substring.

Example

Listing 7.313 shows how the `lastParen` property is used.

Listing 7.313 *Example of the `lastParen` Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
```



```
// Define a regular expression.
exp = new RegExp("(please)", "g");

// Create a string object.
str = "Will you (please) stop yelling!";
myArray = exp.exec(str);

// Inform user what the lastParen property is.
document.write("The RegExp.lastParen is: " + "<b>"
               + RegExp.lastParen + "</b>");

// -->
</script>

</body>
</html>
```

RegExp.leftContext

JavaScript 1.2+, JScript 5.5+
Nav4+, NES3+, IE4+

Syntax

regexp.leftContext

Description

The leftContext property of the RegExp object represents the substring preceding the most recent pattern match.

Example

Listing 7.314 shows how the leftContext property is used. A RegExp pattern is used to get all the contents of the string before the is pattern.

Listing 7.314 *Example of the leftContext Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Define a regular expression pattern and match globally.
pat = /is*/g;

// Create a string object.
var str = "I know where the fish is tonight.";

// Create an array to hold the results.
myArray = pat.exec(str);
```

Listing 7.314 Continued

```
document.write("In the string: " + "<b>" +  
    "I know where the fish is tonight" + "</b><br><br>");  
document.write("The RegExp.leftContext is: " + RegExp.leftContext);  
// -->  
</script>  
  
</body>  
</html>
```

RegExp.multiline

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

regexp.multiline

Description

The multiline property of the RegExp object is used to determine whether pattern matching should be performed across multiple lines.

Example

Listing 7.315 shows how multiline is used.

Listing 7.315 Example of multiline

```
<html>  
<body>  
  
<script type="text/javascript" language="JavaScript">  
<!--  
// Function creates a new regular expression and  
// then executes it against the text in the textarea.  
// Outputs an alert message indicating the boolean value  
// of the RegExp.multiline property.  
function getinfo(){  
  
    var myPat = new RegExp("the", "i");  
    var str = document.form1.mytext.value;  
    myArray = myPat.exec(str);  
  
    alert("RegExp.multiline is: " + RegExp.multiline);  
}  
  
// -->  
</script>
```

```

<form name="form1">
When the text in the text box is changed, and the document is clicked,
an alert box will be displayed showing the value of RegExp.multiline.
<br><br>
<textarea name="mytext" cols="60" rows="8" onChange='getinfo() '>
This is a sample textarea containing some dummy text for
testing purposes. The text in this box will be used to
demonstrate how the multiline property is used. If multiple lines are
read, then RegExp.multiline will be true.
</textarea>
<br>
</form>

</body>
</html>

```

RegExp.prototype

JavaScript 1.1+

Nav3+, NES2+, Opera4+

Syntax

`regex.prototype.property`

`regex.prototype.method`

Description

The prototype property of the RegExp object allows you to add properties or methods to all instances of this class. After properties or methods have been added, any future instances of the object will contain the newly created prototype property.

Example

Listing 7.316 shows how the prototype property is used to create a new property named *myProp*. Then the property is assigned a value and displayed.

Listing 7.316 Example of the prototype Property for RegExp Object

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new property called myProp.
RegExp.myProp = null;

// Create a new RegExp object that contains the new property.
var myExp = new RegExp("now");
myExp.myProp = "nothing special";

```

Listing 7.316 Continued

```
document.write("The value of myProp is: " + myExp.myProp);  
// -->  
</script>  
  
</body>  
</html>
```

RegExp.rightContext

JavaScript 1.2+, JScript 5.5+
Nav4+, NES3+, IE4+

Syntax

regexp.rightContext

Description

The `rightContext` property of the `RegExp` object represents the substring following the most recent pattern match.

Example

Listing 7.317 shows how the `rightContext` property is used. A pattern is defined so that all the contents of the string that appear after the `be` pattern will be displayed.

Listing 7.317 Example of rightContext

```
<html>  
<body>  
  
<script type="text/javascript" language="JavaScript">  
<!--  
// Define a regular expression pattern and match globally.  
pat = /be*/gi;  
  
// Create a string object.  
var str = "Eat Drink and be Merry.";   
  
// Create an array to hold the results.  
myArray = pat.exec(str);  
  
document.write("In the string: " + "<b>" +  
    "Eat Drink and be Merry" + "</b><br><br>");  
document.write("The RegExp.rightContext is: " + RegExp.rightContext);  
  
// -->  
</script>  
  
</body>  
</html>
```

RegExp.source

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

`regexp.source`

Description

The source property of the RegExp object represents the text of the pattern being used for pattern matching.

Example:

Listing 7.318 shows how the source property can be used to get the pattern being matched.

Listing 7.318 *Example of the* source *Property*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new RegExp object.
exp = new RegExp("am", "g");

// Create a string.
str = "This is just a sample sentence.";

myArray = exp.exec(str);

document.write("The source is: " + "<b>" + exp.source + "</b>");
// -->
</script>

</body>
</html>
```

RegExp.test()

JavaScript 1.2+
Nav4+, NES3+, IE4+

Syntax

`regexp.test()`

Description

The `test()` method of the `RegExp` object is used to test for a pattern match in a string. Returns boolean value `true` or `false`.

Example

Listing 7.319 shows how the `test()` method is used.

Listing 7.319 *Example of the `test()` Method*

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Create a new regular expression.
myExp = new RegExp("hope", "g");

// Define a string object.
str = "I hope everything is going well.";

// Test to see if the regular expression exists in the string.
if(myExp.test(str)){
    document.write("The test found \"hope\" in the string: "
        + "<b>" + " I hope everything is going well" + "</b>");
}

// -->
</script>

</body>
</html>
```

RegExp.toSource()

JavaScript 1.3+, ECMAScript 1E+
Nav4.06+

Syntax

```
regexp.toSource()
```

Description

The `toSource()` method of the `Number` object is used to get a string representation of the `Number` object.

Example

Listing 7.320 shows how the `toSource()` method is used.

Listing 7.320 Example of the `toSource()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Creates a new RegExp object and then gets the string
// representation of that object.
var myExp = RegExp("the");
document.write(myExp.toSource());
// -->
</script>

</body>
</html>

```

RegExp.toString()

JavaScript 1.1+, ECMAScript 1E+
Nav3+, NES2+, IE4+, Opera3+

Syntax

```
regexp.toString()
```

Description

The `toString()` method of the `RegExp` object is used to get a string representation of the `RegExp` object.

Example

Listing 7.321 shows how the `toString()` method is used.

Listing 7.321 Example of the `toString()` Method

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Create a new RegExp object.
var myExp = RegExp("and");

// Display the string representation of myExp.
document.write("The RegExp toString output is: " + "<b>" + myExp.toString() +
"</b>");
// -->
</script>

</body>
</html>

```

RegExp.unwatch()

JavaScript 1.2+

Nav4+, NES3+

Syntax

```
regexp.unwatch(prop)
```

Description

The `unwatch()` method of the `RegExp` object will remove a watch point on a property set by the `watch()` method.

Example

Listing 7.322 shows an example for the `unwatch()` method. A property is created using the `prototype()` method. Then a watch point is placed on the newly created property and a message is displayed when the watch point is hit. When the `unwatch()` method is called, the watch point is removed.

Listing 7.322 Example of the `unwatch()` Method

```
<html>
<body>

<script "type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln("myExp." + id + " changed from " + "<b>" + oldValue +
➡ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

myExp = new RegExp("and");

// Create new property p.
RegExp.prototype.p = "the";

// Set the watchpoint on p.
myExp.watch("p",alertme);

myExp.p = "or";
// Remove the watchpoint on p.
myExp.unwatch("p");
myExp.p = "cat";
myExp.p = "dog";
```



```
// Set the watchpoint on p again.
myExp.watch("p",alertme);
myExp.p = "cow";

// -->
</script>
</body>
</html>
```

RegExp.valueOf()

JavaScript 1.1+
Nav3+, NES2+, IE4+

Syntax

regexp.valueOf()

Description

The `valueOf()` method of the `RegExp` object is used to get the primitive value of a `RegExp` object as a number data type.

Example

Listing 7.323 shows an example for the `valueOf()` method. A `RegExp` object is created and set to `myExp`. The document then outputs the result of performing a `valueOf` function on the `RegExp` object.

Listing 7.323 Example of the `RegExp.valueOf()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// Create a new number object.
myExp = new RegExp("or");

// Output the valueOf result.
document.write("The value of myExp is: " + myExp.valueOf());

// -->
</script>

</body>
</html>
```

RegExp.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

regexp.watch(*prop*, *handler*)

Description

The watch() method of the RegExp object will watch for an assignment to a specific property. When an assignment is made, a specified handler will be called to perform a user-defined operation.

Example

Listing 7.324 shows an example for the watch() method. A new property is created using the prototype method. Then a watch point is set on the newly created property. When an assignment is made, the alertme function will be called to display a message.

Listing 7.324 Example of the watch() Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln("myExp." + id + " changed from " + "<b>" +
➡oldValue + "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

myExp = new RegExp("and");

// Create a new property p.
RegExp.prototype.p = "the";

// Set a watchpoint on p.
myExp.watch("p",alertme);

myExp.p = "or";

// -->
</script> type="text

</body>
</html>
```

return

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

return

Description

The return keyword will exit the existing function and return a value.

Example

Listing 7.325 shows an example of using the return statement to return the value of the processing from the function.

Listing 7.325 *Example of* return

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
//Function to get the value of 4 times 3.
function getValue(){
    var myValue = 4*3;
    return myValue;
}

// Function calls getValue() and fills the tmp input box element
// with the value returned.
function fill(){
    var x = getValue();
    document.form1.tmp.value = x;
}
// -->
</script>

<form name="form1">
Value: <input type="text" Name="tmp" Size="5">
<br>
<br>
<input type="button" name="get" value="Get Returned Value" onClick='fill() '>
<br>
<br>
</form>

</body>
</html>
```

ScriptEngine

JScript 2.0+

Syntax

ScriptEngine()

Description

The ScriptEngine() function has three possible return values: JScript, VBScript, and VBA. When implemented in JavaScript scripts, this function returns JScript.

Example

Listing 7.326 prints the complete version information for the Internet Explorer browser interpreting the script. In addition to the ScriptEngine property, it also uses other Internet Explorer specific functions.

Listing 7.326 Using the ScriptEngine Function to Retrieve Information About the Version of the Scripting Engine in an Internet Explorer Browser

```
<script type="text/javascript">
<!--

// Write the scripting engine type.
document.write(ScriptEngine());

// Write the "major" version value to the page.
document.write(" " + ScriptEngineMajorVersion() + ".");

// Write the "minor" version value to the page.
document.write(ScriptEngineMinorVersion());

// Write the build number to the page.
document.write(" build " + ScriptEngineBuildVersion());

//-->
</script>
```

ScriptEngineBuildVersion

JScript 2.0+

Syntax

ScriptEngineBuildVersion

Description

The ScriptEngineBuildVersion() function contains the actual build number of the scripting engine contained on the user's machine.

Example

Listing 7.327 prints the build number of the scripting engine interpreting the script.

Listing 7.327 Using the ScriptEngineBuildVersion Function to Retrieve the Build Number of the Scripting Engine in an Internet Explorer Browser

```
<script type="text/jscript">
<!--

// Write the build number to the page
document.write("Build " + ScriptEngineBuildVersion());

//-->
</script>
```

ScriptEngineMajorVersion

JScript 2.0+

Syntax

```
ScriptEngineMajorVersion()
```

Description

The ScriptEngineMajorVersion() function contains the actual major version number of the scripting engine contained on the user's machine.

Example

Listing 7.328 prints the major version number of the scripting engine interpreting the script.

Listing 7.328 Using the ScriptEngineMajorVersion Function to Retrieve the Major Version Number of the Scripting Engine in an Internet Explorer Browser

```
<script type="text/jscript">
<!--

// Write the build number to the page
document.write("Major Version: " + ScriptEngineMajorVersion());

//-->
</script>
```

ScriptEngineMinorVersion

JScript 2.0+

Syntax

ScriptEngineMinorVersion()

Description

The ScriptEngineMinorVersion() function contains the actual minor version number of the scripting engine contained on the user's machine.

Example

Listing 7.329 prints the minor version number of the scripting engine interpreting the script.

Listing 7.329 Using the ScriptEngineMinorVersion Function to Retrieve the Minor Version Number of the Scripting Engine in an Internet Explorer Browser

```
<script type="text/jscript">
<!--

// Write the build number to the page.
document.write("Minor Version: " + ScriptEngineMinorVersion());

//-->
</script>
```

short

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The short keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

static

JavaScript 1.2+
Nav4+, NES3+,

Syntax

Reserved Keyword

Description

The `static` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

String (Function)

JavaScript 1.2+, JScript 3.0+, ECMAScript 1E+
Nav4+, NES3+, IE4+, Opera4+

Syntax

`String (var)`

Description

The `String()` function is a top-level function, which is often of the `Global` object. It converts the value of any `var` into a readable string.

Example

In Listing 7.330, we will create an instance of the `Date` object, and then use the `String()` function to write out a readable string version of the date.

Listing 7.330 Using the String() Function

```
<script type="text/javascript" language="JavaScript1.2">
<!--
```

```
// Create an instance of Date object.
myDate = new Date (430057843027);
```

```
// writes Thu Aug 18 07:30:43 EST 1983
document.write(String(myDate));
```

```
//-->
</script>
```

String (Object)

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`var variable = new String(string)    JavaScript1.1+`

`"string"`

Description

The `String` object is one of the core JavaScript objects. Instances are created when a program constructs an instance using the `new` keyword and passing it the `String` object. In JavaScript 1.0, instances were also created when programmers quoted characters in their script. Table 7.44 lists the properties and methods used by this object.

Table 7.44 *Properties and Methods Used by the string Object*

Type	Item	Description
Method	<code>anchor()</code>	Creates an instance of the <code><a></code> tag with the <code>name</code> attribute set to the string passed to the method.
	<code>big()</code>	Converts the string into an instance of the <code><big></code> tag.
	<code>blink()</code>	Converts the string into an instance of the <code><blink></code> tag.
	<code>bold()</code>	Converts the string into an instance of the <code></code> tag.
	<code>charAt()</code>	Returns the character at the index passed to the method.
	<code>charCodeAt()</code>	Returns the ISO-Latin-1 number of the character at the index passed to the method.
	<code>concat()</code>	Concatenates the two strings passed to return a new string. This method was added in JavaScript 1.2.
	<code>fixed()</code>	Converts the string into an instance of the <code><tt></code> , fixed pitch font tag.
	<code>fontcolor()</code>	Sets the <code>color</code> attribute of an instance of the <code></code> tag.
	<code>fontSize()</code>	Sets the <code>size</code> attribute of an instance of the <code></code> tag.
	<code>fromCharCode()</code>	Returns the string value of the ISO-Latin-1 number passed to the method.
	<code>indexOf()</code>	Returns the index of the first occurrence of the string passed to the method within an instance of a <code>String</code> object.
	<code>italics()</code>	Converts the string into an instance of the <code><i></code> tag.
	<code>lastIndexOf()</code>	Returns the index of the last occurrence of the string passed to the method within an instance of a <code>String</code> object.
	<code>link()</code>	Converts the string into an instance of the <code><a></code> tag and sets the <code>href</code> attribute with the URL that is passed to the method.

Type	Item	Description
	<code>match()</code>	Returns an array containing the matches found based on the regular expression passed to the method. This method was added in JavaScript 1.2.
	<code>replace()</code>	Performs a search and replace, using the regular expression and replace string passed to the method, on the instance of a <code>String</code> that calls it. This method was added in JavaScript 1.2.
	<code>search()</code>	Returns the index location of the match found in the string passed to the method. A <code>-1</code> is returned if the string is not found. This method was added in JavaScript 1.2.
	<code>slice()</code>	Returns the string between the beginning and ending index passed to the method. If a negative number is passed, the index is referenced from the end of the string passed. This method was added in JavaScript 1.2.
	<code>small()</code>	Converts the string into an instance of the <code><small></code> tag.
	<code>split()</code>	Returns the string split into segments defined by the string and instance limit passed to the method. This method was added in JavaScript 1.1.
	<code>strike()</code>	Converts the string into an instance of the <code><strike></code> tag.
	<code>sub()</code>	Converts the string into an instance of the <code><sub></code> tag.
	<code>substr()</code>	Returns the string beginning with the indexed location and number of characters to return. If a negative number is passed, the index is referenced from the end of the string passed. This method was added in JavaScript 1.2.
	<code>substring()</code>	Returns the string between the beginning and ending index passed to the method.
	<code>sup()</code>	Converts the string into an instance of the <code><sup></code> tag.
	<code>toLocaleLowerCase()</code>	Converts all the characters in the string to lowercase according to the host machine's current locale. This method was added in JavaScript 1.5.

Table 7.44 Continued

Type	Item	Description
Property	toLocaleUpperCase()	Converts all the characters in the string to uppercase according to the host machine's current locale. This method was added in JavaScript 1.5.
	toLowerCase()	Converts all the characters in the string to lowercase.
	toSource()	Returns the string representation of the string passed. This method was added in JavaScript 1.3.
	toString()	Returns the characters passed as type string. This method was added in JavaScript 1.3.
	toUpperCase()	Converts all the characters in the string to uppercase.
	unwatch()	Turns off the watch for a particular property.
	watch()	Turns on the watch for a particular property.
	length	Returns the length of the string.
	prototype	Provides the ability for a programmer to add properties to instances of the String object. This property was added in JavaScript 1.1.

Example

Listing 7.331 displays the use of some of the String properties and methods. It contains a single button. After the user clicks the button, a second window is opened. Various methods are called by a string instance created in the script. The results of such are displayed in the pop-up window.

Listing 7.331 Examples of an Instance of the String Object

```
<html>
<head>
  <title>Examples of the String Object</title>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

  // Open a window to store the results and create a new String object.
  var myWin = open("", "", "width=450,height=200");
  var myString = new String("Hello, World!");

  // Call various methods on this instance and write their results to the
  // window.
  myWin.document.write("Original String, " + myString);
```

```

myWin.document.write(" has " + myString.length + " characters.<br>");
myWin.document.write("Big: " + myString.big() + "<br>");
myWin.document.write("Small: " + myString.small() + "<br>");
myWin.document.write("Blinking: " + myString.blink() + "<br>");
myWin.document.write("Italics: " + myString.italics() + "<br>");
myWin.document.write("Convert to Lower: " + myString.toLowerCase());
myWin.document.write("<br>");
myWin.document.write("Convert to Upper: " + myString.toUpperCase());
myWin.document.write("<br>");

// Close the stream to the window.
myWin.document.close();
}
//-->
</script>
</head>
<body>
<form name="myForm">
  <input type="button" value="Click to Process" name="myButton"
    onClick="openWin()">
</form>
</body>
</html>

```

String.anchor()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

string.anchor(name)

Description

The `anchor()` method will convert the string it is called on to an instance of the `<a>` tag, setting the name attribute to the *name* that is passed.

Example

Listing 7.332 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<a name="HELLO">Hello, World!</a>
```

Listing 7.332 Using the `anchor()` Method of the `String` Object

```

<script type="text/javascript" language="JavaScript1.1">
<!--

```

Listing 7.332 Continued

```
// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the anchor() method on it.
document.write(myString.anchor("HELLO"));

// Close the stream to the window.
document.close();

//-->
</script>
```

String.big()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.big()
```

Description

The `big()` method will convert the string it is called on to an instance of the `<big>` tag.

Example

Listing 7.333 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<big>Hello, World!</big>
```

Listing 7.333 Using the `big()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the big() method on it.
document.write(myString.big());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.blink()

JavaScript 1.0+
Nav2+, NES2+

Syntax

string.blink()

Description

The `blink()` method will convert the string it is called on to an instance of the `<blink>` tag. This method is only supported in Netscape Navigator because it is the only browser that has an implementation of the `<blink>` tag.

Example

Listing 7.334 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<blink>Hello, World!</blink>
```

Listing 7.334 Using the `blink()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">  
<!--
```

```
// Create an instance of the String object.  
var myString = new String("Hello, World!");
```

```
// Write the string to the page after invoking the blink() method on it.  
document.write(myString.blink());
```

```
// Close the stream to the window.  
document.close();
```

```
//-->  
</script>
```

String.bold()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

string.bold()

Description

The `bold()` method will convert the string it is called on to an instance of the `<bold>` tag.

Example

Listing 7.335 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<bold>Hello, World!</bold>
```

Listing 7.335 *Using the `bold()` Method of the `String` Object*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the bold() method on it.
document.write(myString.bold());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.charAt()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`string.charAt(num)`

Description

The `charAt()` method of an instance of the `String` object returns the character located at the indexed, *num*, position passed. This indexing is done from left to right starting with the 0 (zero) position. If the *num* passed is not a valid index in the string, `-1` is returned.

Example

Listing 7.336 creates an instance of a `String` object. When the page is loaded, the user is prompted for an index number. After entering the index number and clicking OK, the character at that indexed location is written to the document. Notice that there is also a check to see whether the character at that location is a space.

Listing 7.336 *Using the `charAt()` Method to Retrieve a Character at a User-specified Location in a String*

```
<html>
<head>
```

```
<title>Using the String.charAt() method</title>
</head>
<body>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Here is a short sentence.");

// Prompt the user for a number.
var myIndex = prompt("Please enter a number", "");

// Store the character at that location in a variable.
var myChar = myString.charAt(myIndex);

// Write the character to the page, but check to see if it
// is a space first.
document.write('<b>The string you searched through was: </b>' + myString);
document.write('<br>The ' + myIndex + ' character in this string is ');

if (myChar == " "){
    document.write('&lt;space&gt;');
}else{
    document.write(myChar);
}

// Close the stream to the window.
myWin.document.close();

<!-->
</script>
</body>
</html>
```

String.charCodeAt()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

string.charCodeAt(*num*)

Description

The `charCodeAt()` method of an instance of the `String` object returns the ISO-Latin-1 number of the character located at the indexed, *num*, position passed. This indexing is done from left to right starting with the 0 (zero) position. If the *num* passed is not a valid index in the string, -1 is returned.

Example

Listing 7.337 creates an instance of a `String` object. When the page is loaded, the user is then prompted for an index number. After entering the index number and clicking OK, the ISO-Latin-1 number of the character at that indexed location is written to the document. Notice that there is also a check to see whether the character at that location is a space.

Listing 7.337 Using the `charCodeAt()` Method to Retrieve a Character at a User-specified Location in a String

```
<html>
<head>
  <title>Using the String.charCodeAt() method</title>
</head>
<body>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Here is a short sentence.");

// Prompt the user for a number.
var myIndex = prompt("Please enter a number", "");

// Store the character code at that location in a variable.
var myCharCode = myString.charCodeAt(myIndex);
var myChar = myString.charAt(myIndex);

// Write the character code to the page.
document.write('<b>The string you searched through was: </b>' + myString);
document.write('<br>The ' + myIndex + ' character in this string is ');

// Check to see if it is a space.
if (myChar == " "){
  document.write('&lt;space>');
}else{
  document.write(myChar);
}

// Write the character code.
document.write(' and its ISO-Latin-1 code is ' + myCharCode);

// Close the stream to the window.
myWin.document.close();

-->
</script>
</body>
</html>
```


String.concat()

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

string.concat(*string2*)

Description

The concat() method of an instance of the String object concatenates the string in *string2* to the end of *string* to return a new string.

Example

Listing 7.338 creates two instances of the String object and uses the concat() method to concatenate them to create a new string. The string is then displayed in an alert box.

Listing 7.338 Using the concat() Method to Concatenate Two Strings

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Create 2 instances of the String object and concatenate
// them together.
var myString1 = new String("Hello, ");
var myString2 = new String("World!");
var myConcatString = myString1.concat(myString2);

// Popup an alert box showing the concatenation.
alert(myConcatString);

//-->
</script>
```

String.constructor

JavaScript 1.1+, JScript 3.0+, ECMAScript 1E+
Nav3+, NES2+, IE4+, Opera3+

Syntax

string.constructor

Description

The constructor property of the String object specifies the function that creates the object.

Example

Listing 7.339 shows an example of the constructor property, which is used to check the type of variable.

Listing 7.339 Example of the constructor *Property*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

if(myString.constructor == String){
    document.write("Object created");
}

//-->
</script>
```

String.fixed()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.fixed()
```

Description

The `fixed()` method will convert the string it is called on to an instance of the `<tt>` tag.

Example

Listing 7.340 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<tt>Hello, World!</tt>
```

Listing 7.340 Using the `fixed()` Method of the *String Object*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the fixed() method on it.
document.write(myString.fixed());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.fontcolor()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.fontcolor (hexnum)
string.fontcolor(color)
```

Description

The fontcolor() method sets the color attribute of an instance of the tag, which it creates. This attribute can either be passed as the hexadecimal equivalent of the color or the actual string that represents that color.

Example

Listing 7.341 creates an instance of the String object and uses the document.write() method to write two instances of the tag to the page. The results of running this script will be the following:

```
Hex usage: <font color="#FF0000">Hello, World!</font>
<br>Color usage: <font color="blue">Hello, World!</font>
```

Listing 7.341 Using the fontcolor() Method of the String Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string twice to the page after invoking the
// fontcolor() method on them.
document.write("Hex usage: " + myString.fontcolor('FF0000'));
document.write("<br>Color usage: " + myString.fontcolor('blue'));

// Close the stream to the window.
document.close();

//-->
</script>
```

String.fontsize()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.fontsize(num)
string.fontsize(string2)
```

Description

The `fontSize()` method sets the size attribute of an instance of the `<font>` tag, which it creates. This attribute can be a number between 1 and 7. If you pass the method the number in the form of a string, the size displayed is relative to the `<basefont>` tag.

Example

Listing 7.342 creates an instance of the `String` object and uses the `document.write()` method to write two instances of the tag to the page. The results of running this script will be the following:

```
Hex usage: <font size="6">Hello, World!</font>
<br>Color usage: <font size="-2">Hello, World!</font>
```

Listing 7.342 *Using the `fontSize()` Method of the `String` Object*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string twice to the page after invoking the
// fontSize() method on them.
document.write("Size=6: " + myString.fontSize(6));
document.write("<br>Size=-2: " + myString.fontSize('-2'));

// Close the stream to the window.
document.close();

//-->
</script>
```

String.fromCharCode()

JavaScript 1.2+, JScript 3.0+, ECMAScript 1E+
Nav4+, NES3+, IE4+

Syntax

```
String.fromCharCode(num1, num2, ..., numN)
String.fromCharCode(keyevent.which)
```

Description

The `fromCharCode()` method of the `String` object returns the characters that correspond to the ISO-Latin-1 numbers (*num1*, *num2*, ..., *numN*) position passed. You can also pass the method a key event and use the `which` property to determine which key has been pressed. The possible key events are `KeyDown`, `KeyPress`, and `KeyUp`.

As you can see in the syntax definition, this is a method of the actual `String` object and not an instance of this object. Because of this, you might want to store the results generated by this method into a variable for future processing.

Example

Listing 7.343 invokes the `fromCharCode()` method on the numbers 88, 89, and 90. The results of this processing are then written to the user's page.

Listing 7.343 *Using the `fromCharCode()` Method to Determine the Characters of the ISO-Latin-1 Numbers Passed*

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Invoke the fromCharCode() method and store the results in
// a variable.
var myString = String.fromCharCode(88,89,90);

// Write the results to the page.
document.write("These numbers evaluate to: " + myString);

// Close the stream to the page.
document.close();

//-->
</script>
```

String.indexOf()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.indexOf(string, num)
string.indexOf(string)
```

Description

The `indexOf()` method of an instance of the `String` object returns the indexed start position of the *string* passed. Additionally, you can specify an index, defined by *num* in the syntax definition, to start your search for the string specified. This method is the same as the `String.lastIndexOf()` method, but it starts at the beginning of the string.

Example

Listing 7.344 creates a simple instance of the `String` object. This instance is then passed to the `indexOf()` method on two occasions with the result written to the user's page. The first occasion looks for a space in the string, which returns 6. The second occasion starts the search at the fourth position, so it returns the location of the letter "l" in the word "world".

Listing 7.344 Using the `indexOf()` Method to Find the Location of a Character in a String

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Look for the first instance of a space.
document.write(myString.indexOf(" ") + '<br>');

// By specifying an indexed location to start looking you
// can return the indexed location of the third instance of
// the letter 'l'.
document.write(myString.indexOf("l", 4));

// Close the stream to the page.
document.close();

//-->
</script>
```

String.italics()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`string.italics()`

Description

The `italics()` method will convert the string it is called on to an instance of the `<i>` tag.

Example

Listing 7.345 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<i>Hello, World!</i>
```

Listing 7.345 Using the `italics()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");
```

```
// Write the string to the page after invoking the italics() method on it.
document.write(myString.italics());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.lastIndexOf()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.lastIndexOf(string, num)
string.lastIndexOf(string)
```

Description

The `lastIndexOf()` method of an instance of the `String` object returns the indexed start position of the *string* passed, starting from the right and going left. Additionally, you can specify an index, defined by *num* in the syntax definition, to start your search for the string specified. This method is the same as the `String.indexOf()` method, but it starts at the end of the string.

Example

Listing 7.346 creates a simple instance of the `String` object. This instance is then passed to the `lastIndexOf()` method on two occasions with the result written to the user's page. The first occasion looks for the last occurrence of the letter `e` in the string, which returns 16. The second occasion starts the search at the third position, so it returns the location of the first `l` in the word "Hello".

Listing 7.346 Using the `lastIndexOf()` Method to Find the Location of a Character in a String

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello World, here I am!");

// Look for the last instance of the letter 'e'.
document.write(myString.lastIndexOf("e") + '<br>');

// By specifying an indexed location to start looking, you
// can return the indexed location of the first instance of
// the letter 'l'.
document.write(myString.lastIndexOf("l", 3));
```

Listing 7.346 Continued

```
// Close the stream to the page
document.close();

//-->
</script>
```

String.length

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`string.length`

Description

The length property of an instance of the String object returns the total length of the string.

Example

Listing 7.347 creates three instances of the String object. The length property of each of these instances is accessed and written to the user's page.

Listing 7.347 Accessing the length Property of an Instance of the String Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString1 = new String("Hello, World");
var myString2 = new String("Here is a longer string");
var myString3 = new String("Here is an even longer string");

// Write the lengths of these strings to the user's page.
document.write(myString1 + ": is " + myString1.length);
document.write(" characters long.<br>");
document.write(myString2 + ": is " + myString2.length);
document.write(" characters long.<br>");
document.write(myString3 + ": is " + myString3.length);
document.write(" characters long.<br>");

// Close the stream to the page.
document.close();

//-->
</script>
```


String.link()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.link(URL)
```

Description

The `link()` method will convert the string it is called on to an instance of the `<a>` tag, setting the `href` attribute to the URL that is passed.

Example

Listing 7.348 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<a href="http://www.purejavascript.com">The online book!</a>
```

Listing 7.348 Using the `link()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("The online book!");

// Write the string to the page after invoking the link() method on it.
document.write(myString.link('http://www.purejavascript.com'));

// Close the stream to the window.
document.close();

//-->
</script>
```

String.localeCompare()

JavaScript 1.5+, ECMAScript 3E+
Nav6

Syntax

```
string.localeCompare (string2)
```

Description

The `localeCompare()` method of an instance of the `String` object compares *string* against *string2*. The resulting numerical value, which can be negative, zero, or positive, orders the strings in a sort order specified by the system default locale.

Example

In Listing 7.349, two strings are created and the `localeCompare()` method is called to compare them.

Listing 7.349 Using the `localeCompare()` Method

```
<script type="text/javascript" language="JavaScript1.5">
<!--

// Define two strings.
myString = new String("hello")
myString2 = new String("world")

// Returns -15 in Netscape 6.
document.write(myString.localeCompare(myString2));

//-->
</script>
```

String.match()

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

`string.match (regexexpression)`

Description

The `match()` method of an instance of the `String` object searches the string in which it is invoked for the regular expression passed to the method. The *regexexpression* is made up of a pattern and flags that dictate what is to be matched. The method returns an array containing the matches found in the string.

TIP

See the reference entry for `RegExp` for a list of the patterns and flags that can be used to create a regular expression.

Example

Listing 7.350 creates an instance of the `String` object and tries to match instances that contain a space followed by some characters. If any matches were returned into the array, they are written to the user's page one at a time.

Listing 7.350 Using the `match()` Method to Match Regular Expressions in a String

```
<script type="text/javascript" language="JavaScript1.2">
<!--
```

```
// Create an instance of the String object and load it with a name.
var myString = new String("Mr. R. Allen Wyke");

// Match occurrences of a space followed by characters.
var myRegExp = /\s\w*/g;
var answerArray = myString.match(myRegExp);

// Check to see if there were any matches found.
if(answerArray == null){
    document.write('No matches were found');
}else{
    document.write('The following matches were found: <br>');

    // Write the contents of the array to the page. This will put
    // R, Allen, and Wyke each on a separate line.
    for(var i = 0; i < answerArray.length; i++){
        document.write(answerArray[i] + '<br>');
    }
}

// Close the stream to the window.
document.close();

//-->
</script>
```

String.prototype

JavaScript 1.1+, ECMAScript 1E+, JScript 3.0+
 Nav3+, NES2+, IE4+, Opera3+

Syntax

String.prototype.property
String.prototype.method

Description

The prototype property of the String object allows a programmer to add properties or methods to a core JavaScript object.

Example

Listing 7.351 creates two instances of the String object. Then it prototypes a new property, type, and a new method, verify(). In the script, the type property is assigned to the string instances and then they are checked using the verify() method. The results of the validation are then written to the user's page.

Listing 7.351 Using the prototype *Property* to Create New Properties and Methods of the String Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Define the method that we prototyped.
function myVerify(){

    // Check to see if the type property we added is set to "Name".
    // If it is, then return true. If not, then return false.
    if(this.type != "Name"){
        return false;
    }else{
        return true;
    }
}

// Create a new property and method of the String object.
String.prototype.type = null;
String.prototype.verify = myVerify;

// Create two instances of the String object and load it with a name.
var myString1 = new String("Mr. R. Allen Wyke");
var myString2 = new String("Mr. Robert J. Wyke");

// Using the prototype we defined, assign the type property to Name
// for the first string and to "Title" for the second.
myString1.type = "Name";
myString2.type = "Title";

// Check each of the types of the strings to see if they are valid.
if(myString1.verify()){
    document.write(myString1 + " has a valid type of " + myString1.type);
}else{
    document.write(myString1 + " has an invalid type of " + myString1.type);
}

document.write('<br>');

if(myString2.verify()){
    document.write(myString2 + " has a valid type of " + myString2.type);
}else{
    document.write(myString2 + " has an invalid type of " + myString2.type);
}

// Close the stream to the window
//document.close();

-->
</script>
```

String.replace()

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

```
string.replace(regexpression, replacestring)
```

Description

The `replace()` method of an instance of the `String` object searches the string in which it is invoked for the regular expression passed to the method. The *regexexpression* is made up of a pattern and flags that dictate what is to be matched. If and when a match is found, the method returns a new string with that match supplanted with the replacement string passed to the method.

TIP

See the reference entry for `RegExp` for a list of the patterns and flags that can be used to create a regular expression.

Example

Listing 7.352 creates an instance of the `String` object. This instance is then searched through to see whether any occurrence of the word "Wyke" is found. If so, it is replaced with "White".

Listing 7.352 *Using the `replace()` Method to Replace Regular Expression Matches in a String*

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Create an instance of the String object and load it with a name.
var myString = new String("Mr. R. Allen Wyke");

// Search for "Wyke" and replace it with "White".
var myRegExp = /Wyke/g;
var newString = myString.replace(myRegExp, "White");

// Write the results to the page.
document.write('Notice the last name in the original string, ' + myString);
document.write(', was replaced and is now ' + newString);

// Close the stream to the window.
document.close();

//-->
</script>
```

String.search()

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

string.search (*regexexpression*)

Description

The `search()` method of an instance of the `String` object searches the string in which it is invoked for the regular expression passed to the method. The *regexexpression* is made up of a pattern and flags that dictate what is to be matched. The method returns the indexed start location of the string if it is found and `-1` if the string does not contain a regular expression match.

TIP

See the reference entry for `RegExp` for a list of the patterns and flags that can be used to create a regular expression.

Example

Listing 7.353 creates an instance of the `String` object, which is searched for the first instance of a space. If a match is found, the indexed start position is returned. The results of running this script are written to the user's page.

Listing 7.353 Using the `search()` Method to Search Regular Expressions in a String

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Create an instance of the String object and load it with a name.
var myString = new String("Mr. R. Allen Wyke");

// Find the first occurrences of a space.
var myRegExp = /\s/;
var answerIdx = myString.search(myRegExp);

// Check to see if there were any matches found.
if(answerIdx == -1){
    document.write('No matches were found');
}else{
    document.write('Your search string was found starting at: ' + answerIdx);
}

// Close the stream to the window.
document.close();
```

```
//-->  
</script>
```

String.slice()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.slice(num1, num2)  
string.slice(num)
```

Description

The `slice()` method of an instance of the `String` object returns the characters in the string between the indexed positions `num1` and `num2` in which the method is invoked. The string itself is zero based, so the first character is in position `0`. It is also possible to pass `num2` as a negative number. In this scenario, the string counts from the end of the string to end the slice.

As the syntax definition states, it is also possible to pass a single index location to the method. In this implementation, the method will not stop at a position and will return all characters until the end of the string.

Example

Listing 7.354 creates an instance of the `String` object. The `slice()` method is invoked on this string and asked to return the first seven characters of the string. The results of running this script are written to the user's page.

Listing 7.354 Using the slice() Method to Return Seven Characters in a String

```
<script type="text/javascript" language="JavaScript1.1">  
!--  
  
// Create an instance of the String object and load it with a name.  
var myString = new String("Mr. R. Allen Wyke");  
  
// Grab the first 7 characters of the string.  
var mySlice = myString.slice(0,6);  
  
// Write the results to the page.  
document.write('The first 7 characters of our string, ' + myString);  
document.write(', are: ' + mySlice);  
  
// Close the stream to the window.  
document.close();  
  
!-->  
</script>
```

String.small()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.small()
```

Description:

The `small()` method will convert the string it is called on to an instance of the `<small>` tag.

Example

Listing 7.355 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<small>Hello, World!</small>
```

Listing 7.355 *Using the `small()` Method of the `String` Object*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the small() method on it.
document.write(myString.small());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.split()

JavaScript 1.1+, JScript 1.0+, ECMAScript 1E+
Nav3+, NES2+, IE3+, Opera3+

Syntax

```
string.split(separator, num)
string.split(separator)
string.split(regexpression, num)
string.split(regexpression)
```


Description

The `split()` method of an instance of the `String` object splits the string in which it is invoked into separate strings based on the *regexpression* or *separator* passed to the method. If a regular expression is passed, it is made up of a pattern and flags that dictate what is to be matched. The *separator* is a string or character that is matched to perform the separation.

NOTE

If the `language` attribute of the `<script>` tag is set to "JavaScript1.2" when using the second syntactical definition and the separator is a space, consecutive spaces are treated differently. In JavaScript 1.1, each space would be split and returned as part of the results. So if there were an instance of three consecutive spaces, two of the spaces would be returned in the array. Specifying JavaScript 1.2 tells the interpreter to treat the three spaces as a single space, so you are able to perform proper splits.

The method returns an array containing each of the segments found in the string.

TIP

See the reference entry for `RegExp` for a list of the patterns and flags that can be used to create a regular expression.

Example

Listing 7.356 creates an instance of the `String` object. This instance is then split, looking for a space as the separator, using each of the syntactical definitions. The results are then written to the user's page.

Listing 7.356 Using the `split()` Method to Split the String Passed into Separate Strings

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Define a function to handle writing the results.
function genResults(arrayName, testName){
    document.write('<b>Currently Evaluating: ' + testName + '</b><hr>');

    // Check to see if there were any spaces found.
    if(arrayName == null){
        document.write('No matches were found');
    }else{

        // Write the contents of the array to the page. This will put
        // R, Allen, and Wyke each on a separate line.
        for(var i = 0; i < arrayName.length; i++){
            document.write('[' + i + ']: ' + arrayName[i] + '<br>');
```

Listing 7.356 Continued

```
    }  
  }  
  document.write('<p>');  
}  
  
// Create an instance of the String object and load it with a name.  
var myString = new String("Mr. R. Allen Wyke");  
  
// Define a regular expression and a separator. Both are set to  
// split on a single space.  
var myRegExp = /\s/g;  
var mySeparator = " ";  
  
genResults(myString.split(mySeparator), "Separator Only");  
genResults(myString.split(mySeparator, 2), "Separator With Limit of 2");  
genResults(myString.split(myRegExp), "Regular Expression Only");  
genResults(myString.split(myRegExp, 3), "Regular Expression With Limit of 3");  
  
// Close the stream to the window.  
document.close();  
  
//-->  
</script>
```

String.strike()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.strike()
```

Description

The `strike()` method will convert the string it is called on to an instance of the `<strike>` tag.

Example

Listing 7.357 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<strike>Hello, World!</strike>
```

Listing 7.357 Using the `strike()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">  
!!--
```

```
// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the strike() method on it.
document.write(myString.strike());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.sub()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

string.sub()

Description

The sub() method will convert the string it is called on to an instance of the <sub> tag.

Example

Listing 7.358 creates an instance of the String object and uses the document.write() method to write the tag to the page. The results of running this script will be the following:

```
<sub>Hello, World!</sub>
```

Listing 7.358 Using the sub() Method of the String Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the sub() method on it.
document.write(myString.sub());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.substr()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.substr(num1, num2)  
string.substr (num)
```

Description

The `substr()` method of an instance of the `String` object returns the characters in the string, starting with the indexed position `num1` and counting to `num2` characters. The string itself is zero based, so the first character is in position `0`. It is also possible to pass `num1` as a negative number. In this scenario, the string starts from the end of the string to begin the substring extraction.

As the syntax definition states, it is also possible to pass a single index location to the method. In this implementation, the method will not stop at a position and will return all characters until the end of the string.

Example

Listing 7.359 creates an instance of the `String` object. The `substr()` method is invoked on this string and asked to return the first six characters of the string. The results of running this script are written to the user's page.

Listing 7.359 Using the `substr()` Method to Return Seven Characters in a String

```
<script type="text/javascript" language="JavaScript1.1">  
!--  
  
// Create an instance of the String object and load it with a name.  
var myString = new String("Mr. R. Allen Wyke");  
  
// Grab the first 7 characters of the string.  
var mySubString = myString.substr(0,6);  
  
// Write the results to the page.  
document.write('The first 6 characters of our string, ' + myString);  
document.write(', are: ' + mySubString);  
  
// Close the stream to the window.  
document.close();  
  
-->  
</script>
```

String.substring()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.substring(num1, num2)  
string.substring(num)
```

Description

The `substring()` method of an instance of the `String` object returns the characters in the string, starting with the indexed position `num1` and ending with the character before `num2`. The string itself is zero based, so the first character is in position 0.

If you pass `num1` as a negative number, it will be treated as 0. Likewise if you pass `num2` as a value greater than the `String.length` property, it will be treated as `String.length`. And finally, if `num1` equals `num2`, an empty string is returned.

As the syntax definition states, it is also possible to pass a single index location to the method. In this implementation, the method will not stop at a position and will return all characters until the end of the string.

NOTE

If the language attribute of the `<script>` tag is set to "JavaScript1.2" and a Navigator 4 browser is interpreting the script, a runtime out of memory error will be produced if `num1` is greater than `num2`. Without this attribute set, the method returns a substring beginning with `num2` and ending with `num1 - 1`.

Example

Listing 7.360 creates an instance of the `String` object. The `substring()` method is invoked on this string and asked to return the first 10 characters of the string. The results of running this script are written to the user's page.

Listing 7.360 Using the `substring()` Method to Return 11 Characters in a String

```
<script type="text/javascript" language="JavaScript1.1">  
<!--  
  
// Create an instance of the String object and load it with a name.  
var myString = new String("Mr. R. Allen Wyke");  
  
// Grab the first 11 characters of the string.  
var mySubString = myString.substring(0,10);  
  
// Write the results to the page.  
document.write('The first 10 characters of our string, ' + myString);  
document.write(', are: ' + mySubString);
```

Listing 7.360 Continued

```
// Close the stream to the window.
document.close();

//-->
</script>
```

String.sup()

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.sup()
```

Description

The `sup()` method will convert the string it is called on to an instance of the `<sup>` tag.

Example

Listing 7.361 creates an instance of the `String` object and uses the `document.write()` method to write the tag to the page. The results of running this script will be the following:

```
<sup>Hello, World!</sup>
```

Listing 7.361 Using the `sup()` Method of the `String` Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Hello, World!");

// Write the string to the page after invoking the sup() method on it.
document.write(myString.sup());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.toLocaleLowerCase()

JavaScript 1.5+, JScript 5.5+, ECMAScript 3E+
Nav6+, IE5.5+

Syntax

```
string.toLocaleLowerCase()
```

Description

The `toLocaleLowerCase()` method of an instance of a `String` object converts the characters in that string to all lowercase values. This is often used when a programmer tries to evaluate a string a user has entered and does not care about case. This method differs from the `toLowerCase()` method by yielding its result to the host environment's current locale.

Example

Listing 7.362 pops up a prompt box and asks the user to enter various case text. After the user clicks OK, the lowercase version of the string is written to the page.

Listing 7.362 Using the `toLocaleLowerCase()` Method of the `String` Object to Convert a String Entered by a User to Lowercase

```
<script type="text/javascript" language="JavaScript1.5">
<!--

// Create an instance of the String object.
var myString = new String(prompt("Please enter some various case text", ""));

// Convert the text to lowercase and write it to the page.
document.write(myString.toLocaleLowerCase());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.toLocaleUpperCase()

JavaScript 1.5+, JScript 5.5+, ECMAScript 3E+
 Nav6+, IE5.5+

Syntax

```
string.toLocaleUpperCase()
```

Description

The `toLocaleUpperCase()` method of an instance of a `String` object converts the characters in that string to all uppercase values. This is often used when a programmer is trying to evaluate a string a user has entered, and case is not an issue. This method differs from the `toUpperCase()` method by yielding its result to the host environment's current locale.

Example

Listing 7.363 pops up a prompt box and asks the user to enter various case text. After the user clicks OK, the uppercase version of the string is written to the page.

Listing 7.363 Using the `toLocaleUpperCase()` Method of the String Object to Convert a String Entered by a User to Uppercase

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String(prompt("Please enter some various case text", ""));

// Convert the text to uppercase and write it to the page.
document.write(myString.toLocaleUpperCase());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.toLowerCase()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
string.toLowerCase()
```

Description

The `toLowerCase()` method of an instance of a `String` object converts the characters in that string to all lowercase values. This is often used when a programmer tries to evaluate a string a user has entered and does not care about case.

Example

Listing 7.364 pops up a prompt box and asks the user to enter various case text. After the user clicks OK, the lowercase version of the string is written to the page.

Listing 7.364 Using the `toLowerCase()` Method of the String Object to Convert a String Entered by a User to Lowercase

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String(prompt("Please enter some various case text", ""));

// Convert the text to lowercase and write it to the page.
document.write(myString.toLowerCase());

// Close the stream to the window.
document.close();
```



```
//-->
</script>
```

String.toSource()

JavaScript 1.3+, JScript 3.0+, ECMAScript 2E+
Nav4.06+, IE4+

Syntax

```
string.toSource()
String.toSource()
```

Description

The `toSource()` method of the `String` object will return something similar to the following:

```
function String() { [native code] }
```

If it is invoked on an instance of the `String` object, it will contain the source of the instance you created.

Example

Listing 7.365 creates an instance of the `String` object. The `toSource()` method is then applied to the instance and the results are written to the page. A second `document.write()` method writes the results of applying the method to the core object.

The result of this application of the method should be something similar to the following:

```
(new String("Hello, World!"))
```

Listing 7.365 Using the `toSource()` Method of the `String` Object to Obtain the Source of the Object or an Instance of It

```
<script type="text/javascript" language="JavaScript1.3">
<!--
```

```
// Create an instance of the String object.
var myString = new String("Hello, World!");
```

```
// Apply the toSource() method to the instance and the core String
// object.
document.write(myString.toSource() + '<br>');
document.write(String.toSource());
```

```
// Close the stream to the window.
document.close();
```

```
//-->
</script>
```

String.toString()

JavaScript 1.1+, JScript1.0+, ECMAScript 1E+
Nav3+, NES2+, IE3+

Syntax

```
string.toString()  
String.toString()
```

Description

The `toString()` method of the `String` object, if invoked on the core `String` object, will return the object type or the name of the constructor that created the object. This will be something similar to the following:

```
function String() { [native code] }
```

If it is invoked on an instance of the `String` object, it will contain the source string of the instance you created.

Example

Listing 7.366 creates an instance of the `String` object. The `toString()` method is then applied to the instance and the results are written to the page. A second `document.write()` method writes the results of applying the method to the core object.

The result of this application of the method should be something similar to the following:

```
Hello, World!
```

Listing 7.366 Using the toString() Method of the String Object to Obtain the Source of the Object or an Instance of It

```
<script type="text/javascript" language="JavaScript1.3">  
<!--  
  
// Create an instance of the String object.  
var myString = new String("Hello, World!");  
  
// Apply the toString() method to the instance and the core String  
// object.  
document.write(myString.toString() + '<br>');  
document.write(String.toString());  
  
// Close the stream to the window.  
document.close();  
  
//-->  
</script>
```

String.toUpperCase()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

string.toUpperCase ()

Description

The toUpperCase() method of an instance of a String object converts the characters in that string to all uppercase values. This is often used when a programmer is trying to evaluate a string a user has entered, and case is not an issue.

Example

Listing 7.367 pops up a prompt box and asks the user to enter various case text. After the user clicks OK, the uppercase version of the string is written to the page.

Listing 7.367 Using the toUpperCase() Method of the String Object to Convert a String Entered by a User to Uppercase

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String(prompt("Please enter some various case text", ""));

// Convert the text to uppercase and write it to the page.
document.write(myString.toUpperCase());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

string.unwatch (*property*)

Description

The unwatch() method of the String object, is used to turn off the watch on a particular property specified by *property*.

Example

Listing 7.368 shows how the `unwatch()` method is used to stop watching the user-defined property `p`.

Listing 7.368 *Example of* `unwatch()` *Method of the Array Object*

```
<script type="text/javascript">
<!--

// Create a function to handle watch.
function alertme(id, oldValue, newValue){
    document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

// Create instance of string.
var myString = new String();

// Create property for string.
String.prototype.p = 1;

// Watch property value.
myString.watch("p", alertme);

// Change value.
myString.p = 2;

// Turn off watch.
myString.unwatch("p");
myString.p = 3;

// -->
</script>
```

String.valueOf()

JavaScript 1.1+, JScript 3.0+, ECMAScript 1E+
Nav3+, NES2+, IE4+, Opera3+

Syntax

```
string.valueOf()
```

Description

The `valueOf()` method returns the primitive value of the object. In terms of an instance of a `String` object, this method returns the string itself.

Example

In Listing 7.369, an instance of the `String` object is created. Because the `valueOf()` method returns the actual value of the string, "Here is some random text" is written to the user's page.

Listing 7.369 *Using the `valueOf()` Method to Return the Value of the `String()` Instance*

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Create an instance of the String object.
var myString = new String("Here is some random text.");

// Write the value of the string to the page.
document.write('The value of my string instance is: ' + myString.valueOf());

// Close the stream to the window.
document.close();

//-->
</script>
```

String.watch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

```
string.watch(property)
```

Description

The `watch()` method of the `String` object, is used to turn on the watch on a particular property specified by *property*.

Example

Listing 7.370 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 7.370 *Example of `watch()` Method of the Array Object*

```
<script type="text/javascript">
<!--

// Create a function to handle watch.
function alertme(id, oldValue, newValue){
```

Listing 7.370 Continued

```
document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
document.writeln(newValue + "<br>");
return newValue;
}

// Create instance of string.
var myString = new String();

// Create property for string.
String.prototype.p = 1;

// Watch property value.
myString.watch("p", alertme);

// Change value.
myString.p = 2;

// -->
</script>
```

sun

JavaScript 1.1+
Nav3+, NES2+

Syntax

sun

Description

The sun object allows you to access any class within the package sun.*. It is shorter and works the same way as using the Packages.sun property.

Example

Listing 7.371 shows how the sun property can be used to determine whether the current browser supports the sun package if Java is installed.

Listing 7.371 Example Using the sun Object

```
<script type="text/javascript" language="JavaScript1.1">
<!--

// Checks to see if the sun Java package is installed.
if(sun){
    document.write ("This browser has the sun package");
}else{
    document.write ("This browser does not have the sun package");
}
```

```
// -->  
</script>
```

super

JavaScript 1.3+
Nav4.06+, NES3+,

Syntax

Reserved Keyword

Description

The `super` keyword was reserved for future use but has not currently been implemented.

Example

No example can be provided because `super` has not been implemented.

switch

JavaScript 1.2+, JScript 3.0+
Nav4+, NES3+, IE4+

Syntax

```
switch(expression){  
    case label1:  
        code;  
        break;  
    case label2:  
        code;  
        break;  
    case labelN:  
        code;  
        break;  
    default:  
        code;  
}
```

Description

The `switch` statement allows you to process the *expression* passed by matching it with a label—from *label1* to *labelN*. If there is a match, the code following that label is executed. If the *expression* passed does not match a label, the `default` section is executed. Note that you can have as many labels as you deem necessary in your script.

Example

Listing 7.372 has a text field and a button. Users are asked to enter a day of the week into the field. When they press the button, a function is called that contains a `switch`,

which verifies they entered a correct day. If so, an alert box pops up and tells users what part of the week they entered.

Listing 7.372 Using a switch Statement to Process Data

```
<html>
<head>
  <title>Using the switch statement</title>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Display an alert box that contains the value of the
// submit button.
function verifyDay(form){

    // Read the text entered in a variable and convert it to uppercase.
    var myEntry = form.day.value.toUpperCase();

    // Define what you return.
    var firstPart = "You have entered a day at the beginning of the week";
    var endPart = "You have entered a day at the end of the week";
    var weekEnd = "You have entered a weekend day";

    // Use a switch statement to perform your processing.
    switch(myEntry){
        case "MONDAY" :
            alert(firstPart);
            break;
        case "TUESDAY" :
            alert(firstPart);
            break;
        case "WEDNESDAY" :
            alert('You have entered a "hump" day');
            break;
        case "THURSDAY" :
            alert(endPart);
            break;
        case "FRIDAY" :
            alert(endPart);
            break;
        case "SATURDAY" :
            alert(weekEnd);
            break;
        case "SUNDAY" :
            alert(weekEnd);
            break;
        default :
            alert('You have entered an invalid day');
    }
}
```



```
//-->
</script>
</head>
<body>
<form name="myForm">
<b>Please enter a day of the week:</b><br>
  <input type="text" value="" name="day">
  <input type="button" value="Verify" name="myButton"
    onClick='verifyDay(this.form) '>
</form>
</body>
</html>
```

synchronized

JavaScript 1.2+, JScript3.0+
Nav4+, NES3+, IE4+

Syntax

Reserved Keyword

Description

The synchronized keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

this

JavaScript 1.0+, JScript 1.0+
Nav2+, NES2+, IE2+, Opera3+

Syntax

this
this.*property*

Description

The this keyword is used to refer to the current object and is often used to pass entire objects, such as those contained in Form instances, to functions and methods specified in scripts.

Example

Listing 7.373 contains a text box, a text area, and a button. When the user clicks the button, an onClick event handler passes the form's information, using the this keyword, to a function defined in the <head> of the document. The function opens a second, smaller window and writes several properties of the form to the page.

Listing 7.373 Using this to Pass All Form Data to a Function

```
<html>
<head>
  <title>Using this in passing form information</title>
<script type="text/javascript">
<!--

function displayInfo(form){

  // Open a window to store the results.
  var myWin = open("", "", "width=450,height=200");

  // Write the text boxes properties to the window.
  myWin.document.write("The defaultValue of the text box is: ");
  myWin.document.write(form.myText.defaultValue);
  myWin.document.write("<br>The name of the text area is: ");
  myWin.document.write(form.myTextArea.name);
  myWin.document.write("<br>The value of the button is: ");
  myWin.document.write(form.myButton.value);

  // Close the stream to the window.
  myWin.document.close();
}
//-->
</script>
</head>
<body>
<form name="myForm">
  <textarea name="myTextArea" rows=2 cols=50>
    Here is some text in my text area.
  </textarea>
  <br>
  <input type="text" value="Change Me?" name="myText">
  <br>
  <input type="button" value="Display Information" name="myButton"
    onClick='displayInfo(this.form)''>
</form>
</body>
</html>
```

Listing 7.374 creates a `vehicle` object that has three properties: number of doors, color, and the type of vehicle. The `this` keyword is used to associate these properties with the object when it is referenced in other scripts. Further down in the example, an instance of the `vehicle` object is created and its properties are written to the page.

Listing 7.374 Using this to Internally Reference Properties of a User-defined Object

```
<html>
<head>
```

```
<title>Using this in object creation</title>
<script type="text/javascript">
<!--

// Create vehicle object.
function vehicle(nDoors, sColor, sType){

    // Define the characteristics of the vehicle and associate
    // them with a new instance using the "this" keyword.
    this.doors = nDoors;
    this.color = sColor;
    this.type = sType;
}

//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--

// Create an instance of the vehicle.
var myVehicle = new vehicle(4, "red", "Toyota");

// Call the properties of your object using the dot convention
// found throughout JavaScript.
document.writeln("I have created a " + myVehicle.type);
document.writeln(" that is " + myVehicle.color);
document.writeln(" and has " + myVehicle.doors + " doors.");
//-->
</script>
</body>
</html>
```

throw

JavaScript 1.4+, JScript 5.0+, ECMAScript 3E+
Nav6+, NES3+, IE5+

Syntax

throw *exception*

Description

The throw element of the JavaScript language was a reserved keyword in the JavaScript 1.3 and JScript 3.0 languages and in the Netscape Enterprise server 3 environment. Netscape 6 and Internet Explorer 5 were the first browsers to implement the keyword. It is used to generate an error condition handled by a try...catch statement or to pass errors found in these statements to higher-level handlers.

Example

Listing 7.375 contains a text box and button. The user is asked to insert a numeric value into the text box and press the button. When this is done, the `onClick` event handler of the button calls a function to check whether the entered text was numeric. If it is not, the `myErrorHandler` function is called to handle the error. The `myErrorHandler` function contains a `try..catch` statement that allows the programmer to define what happens on an incorrect entry. In this example, an alert box is displayed containing an error code and message.

Listing 7.375 *This Example Uses throw in a try..catch Statement*

```
<html>
<head>
  <title>Using throw in a try..catch statement</title>
<script type="text/javascript" language="JavaScript1.5">
<!--

// Declare a function to handle errors.
function myErrorHandler(data){
  try{
    // Check to see if the value passed is "string" or "NaN"
    // then "throw" the appropriate error.
    if(data == "string"){
      throw "E0";
    }else{
      throw "E1";
    }
  }
  catch(e){

    // If the error thrown was "E0" then return the following.
    if(e == "E0"){
      return("Error (" + e + "): Entry must be numeric.");
    }else{

      // Pass off to a higher level handler.
      return("Error (" + e + "): Entry must be numeric.");
    }
  }
}

// This function reads in the form data and calls the appropriate error.
function processData(form){

  // Check to see if a number was passed.
  if(isNaN(parseInt(form.myText.value))){
    alert(myErrorHandler("string"));
  }else{
```

```
        alert("You have correctly entered a number");
    }
}
//-->
</script>
</head>
<body>
<form name="myForm">
    Please enter a number:
    <input type="text" size="10" value="" name="myText">
    <input type="button" value="Process" name="myButton"
        onClick='processData(this.form) '>
</form>
</body>
</html>
```

throws

JavaScript 1.2+
Nav4, NES3+

Syntax

Reserved Keyword

Description

The throws keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

transient

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The transient keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

true

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `true` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

try...catch...finally

JScript 5.0+, JavaScript 1.4, ECMAScript 3E+
Nav6+, NES3+, IE5+

Syntax

```
try{  
    statement1  
    throw exception;  
}catch(exception if expression){  
    statement2  
    throw exception;  
}catch(exception){  
    statement3  
    throw exception;  
}finally{  
    statement4  
    throw exception;  
}
```

Description

The `try...catch...finally` statement of the JavaScript language contained the `try`, `catch`, and `finally` reserve keywords in the JavaScript 1.3 and JScript 3.0 languages and in the Netscape Enterprise server 3 environments. Internet Explorer 5 and Netscape 6 were the first browsers to implement this feature.

This statement, of which both `catch` and `finally` elements are optional and can contain additional nested `try...catch...finally` statements, can be used to handle all or some of the errors that can occur in a script. If an error is not handled by the statement, it is passed on so other statements can handle the error. If there are no other statements to handle the error, it is passed to the browser to handle. This usually means a pop-up dialog box to the user or writing the information to a hidden error dialog.

As soon as an error occurs, the value thrown is passed to the catch clause via *exception*. If the error cannot be handled, another throw statement is used to pass the error to a higher-level (or next in line) handler if one has been defined.

statement1 is initially where an error can occur. If an if statement is used in the catch clause, *exception* is only caught if *expression* is true, at which time *statement2* is used to handle the error. If it is not handled, *statement3* will be evaluated to handle the exception. As mentioned, if no block can handle the exception, it is passed back to the browser. The finally block, which is optional, is executed before this is done.

NOTE

The ability to have multiple catch clauses is an extension within the Netscape 6 environment.

Example

Listing 7.376 contains a text box and button. The user is asked to insert a numeric value into the text box and click the button. When this is done, the onClick event handler of the button calls a function to check whether the entered text was numeric. If it is not, the myErrorHandler function is called to handle the error. The myErrorHandler function contains a try...catch...finally statement that allows the programmer to define what happens on an incorrect entry. In this example, an alert box is displayed containing an error code and message; and a second alert box is popped up, within the finally block, to signal the end of the error processing.

Two levels of error handling are contained in this example to demonstrate nested usage.

Listing 7.376 This Example Uses a try...catch...finally Statement to Handle an Incorrect Entry

```
<html>
<head>
<script type="text/javascript" language="JavaScript1.5">
<!--

// Declare a function to handle errors.
function myErrorHandler(data){
    try{
        try{
            // Check to see if the value passed is "string" or "NaN"
            // then "throw" the appropriate error.
            if(data == "string"){
                throw "E0";
            }else{
                throw "E1";
            }
        }
    }catch(e){

        // If the error thrown was "E0" then return the following.
        if(e == "E0"){
```

Listing 7.376 Continued

```
        return("Error (" + e + "): Entry must be numeric.");
    }else{

        // Pass off to a higher level handler.
        throw e;
    }
}

// This is the higher level handler for demonstration purposes.
}catch(e){
    return("Error (" + e + "): Entry was invalid.");
}finally{
    alert("Error processing complete!");
}
}

// This function reads in the form data and calls the appropriate error.
function processData(form){

    // Check to see if a number was passed.
    if(isNaN(parseInt(form.myText.value))){
        alert(myErrorHandler("string"));
    }else{
        alert("You have correctly entered a number");
    }
}
//-->
</script>
</head>
<body>
<form name="myForm">
    Please enter a number:
    <input type="text" size="10" value="" name="myText">
    <input type="button" value="Process" name="myButton"
        onClick='processData(this.form)'\>
</form>
</body>
</html>
```

typeof

JavaScript 1.1+, JScript 1.0+
Nav3+, NES2+, IE3+, Opera3+

Syntax

typeof (*variable*)

Description

The `typeof` unary operator is used to determine the type of the variable passed to it. The return values of this operator are `boolean`, `number`, `object`, `string`, or `undefined`.

NOTE

Because `undefined` was not fully implemented until JavaScript 1.3 and JScript 3.0, many supporting browsers return `null` when passing a variable that has not been defined.

Example

Listing 7.377 creates `boolean`, `number`, `object`, `string`, and `undefined` variable instances, and then uses the `typeof` operator to write their types to the page.

Listing 7.377 This Example Uses the `typeof` Unary Operator to Return the Types for Four Different Variables

```
<html>
<head>
  <title>Using typeof to determine the type of variables
</title>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Declare 4 variables of different types.
var bMyVar = true;
var nMyVar = 35;
var sMyVar = "This is a string";
var uMyVar;

//-->
</script>
</head>
<body>
<script type="text/javascript" language="JavaScript1.1">
<!--

// Declare 4 variables of different types.
document.writeln("bMyVar = " + typeof(bMyVar));
document.writeln("<br>nMyVar = " + typeof(nMyVar));
document.writeln("<br>sMyVar = " + typeof(sMyVar));
document.writeln("<br>uMyVar = " + typeof(uMyVar));

//-->
</script>
</body>
</html>
```

undefined

JavaScript 1.3+, JScript 5.0+, ECMAScript 1E+
Nav4.06+, IE5+

Syntax

undefined

Description

The undefined property is a primitive value of the Global object. It is returned by variables that have not had values assigned to them. It is also returned by methods if the variable being evaluated is not assigned a value. Browsers not supporting this property return null on the undefined variables.

NOTE

ECMAScript 1st Edition only defines undefined as a primitive value, and does not define it as a property of the Global object.

Example

Listing 7.378 creates the variable myVariable, and then checks to see whether it is undefined in an if statement. Note that Opera 3 and later browsers, even though they do not officially support this property, return null but evaluate the if statement in this example as true.

Listing 7.378 *Testing a Variable to See Whether It Is* undefined

```
<script type="text/javascript" language="JavaScript1.3">
<!--

// Create a variable.
var myVariable;

// Evaluate the variable in an if statement and write its
// value to the page.
if(myVariable == undefined){
    document.write("This variable is undefined at the moment");
}else{
    document.write("This variables value is: " + myVariable);
}
//-->
</script>
```

unescape()

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

`unescape (string)`

Description

The `unescape()` method takes a hexadecimal value and returns the ISO-Latin-1 ASCII equivalent. This method performs the opposite operation of the `escape()` method and is commonly used to escape user-entered data before form submission.

Example

Listing 7.379 declares a local variable, `escapedVal`, and passes it to the `unescape()` method. The result, "@", is then written to the page.

Listing 7.379 *Using the `unescape()` Method to Convert a Hexadecimal Value to Its ASCII Equivalent*

```
<script type="text/javascript">
<!--

// Create a variable.
var escapedVal = "%40";

// Evaluate the variable and place the value in a variable.
var unescapedVal = unescape(escapedVal);

document.write('The <I>escapedVal</I> value (' + escapedVal + ") ");
document.write("evaluates to " + unescapedVal);

//-->
</script>
```

var

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
var variable
var variable = value
```

Description

The `var` keyword is used to declare variables within a script. If it is used in a function, the scope of the variable is confined to that function. If used outside of a function, it is not limited and can be accessed anywhere on the page.

Example

Listing 7.380 declares the variable, `myVar`, in three different locations. It is declared once at a global level, once within a function where it is the returned value, and once in a function where it is written to the page.

Listing 7.380 *This Example Shows Using the var Keyword in Three Different Instances*

```
<html>
<head>
  <title>Examples of the var Keyword</title>
<script type="text/javascript">
<!--

// Declare a global variable.
var myVar = "Global";

// Declare a variable of the same name in this function
// and return it.
function myFunc(){
  var myVar = "Function";
  return myVar;
}

// Declare a variable of the same name in a second function
// and write it to the page.
function mySecFunc(){
  var myVar = "Second Function";
  document.write("<br>The value of myVar when called by mySecFunc() is: ");
  document.write(myVar);
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--

// Write the value of the global variable.
document.write("The value of myVar when called is: " + myVar)

// Write the value returned by the function.
document.write("<br>The value of myVar when called by myFunc() is: ");
document.write(myFunc());

// Call the second function to write its results.
mySecFunc();

//-->
</script>
</body>
</html>
```

VBAArray

JScript 3.0+

Syntax

```
var variable = new VBAArray(vbarray)
```

Description

The VBAArray object provides access to Visual Basic safeArrays. These arrays are often written on the same HTML page and are written in VBScript. Table 7.45 lists the methods of the VBAArray object.

Table 7.45 *Methods of the VBAArray Object*

Method	Description
dimensions()	Returns the number of dimensions in the array
getItem()	Returns the item at a specified location
lbound()	Returns the lowest index value of the dimension in the array
toArray()	Returns a JScript array from the VBAArray passed
ubound()	Returns the highest index value of the dimension in the array

Example

Listing 7.381 calls a VBScript function from a JScript new operator to create a Visual Basic safe array. The function itself writes the contents of this two-dimensional array to the user's page.

Listing 7.381 *Creating a VBAArray*

```
<script type="text/vbscript">

' Define the VB Array
Function myVBAArray()
    ' Define variables for 2-D array positioning
    Dim i
    Dim j

    ' Define variable to hold incremented values to put into
    ' array and assign it an initial value of 1
    Dim k
    k = 1

    ' Create a 2-D array
    Dim myArray(1, 1)

    ' Iterate through 2-D array and put incremented value in
    For i = 0 To 1
        For j = 0 To 1
            myArray(j, i) = k
```

Listing 7.381 Continued

```
' Write the value to the screen
document.writeln(k)
k = k + 1
Next
document.writeln("<br>")
Next

' Return the array to the calling function
myVBAArray = myArray
End Function

</script>
<script type="text/jscript">
<!--

// Create a new instance of VBAArray
var myArray = new VBAArray(myVBAArray());

//-->
</script>
```

VBAArray.dimensions()

JScript 3.0+

Syntax

```
vbarrray.dimensions()
```

Description

The `dimensions()` method of an instance of a `VBAArray` returns the number of dimensions of the array.

Example

Listing 7.382 calls a VBScript function from a JScript `new` operator to create a Visual Basic safe array. The array itself writes the contents of this two-dimensional array to the user's page. An alert box is also invoked that contains the number of dimensions of the array.

Listing 7.382 Viewing the Number of Dimensions of a VBAArray Object

```
<script type="text/vbscript">

' Define the VB Array
Function myVBAArray()

' Define variables for 2-D array positioning
Dim i
Dim j
```

```

' Define variable to hold incremented values to put into
' array and assign it an initial value of 1
Dim k
k = 1

' Create a 2-D array
Dim myArray(1, 1)
' Iterate through 2-D array and put incremented value in
For i = 0 To 1
    For j = 0 To 1
        myArray(j, i) = k

        ' Write the value to the screen
        document.writeln(k)
        k = k + 1
    Next
    document.writeln("<br>")
Next
' Return the array to the calling function
myVBAArray = myArray
End Function

</script>
<script type="text/javascript">
<!--

// Create a new instance of VBAArray.
var myArray = new VBAArray(myVBAArray());
alert(myArray.dimensions());

//-->
</script>

```

VBAArray.getItem()

JScript 3.0+

Syntax

```

vbarray.getItem(index)
vbarray.getItem(indexA, indexB, ..., indexN)

```

Description

The `getItem()` method of an instance of a `VBAArray` returns the value at the *index* passed. If the array is multidimensional, you pass the necessary coordinates to access the location you want.

Example

Listing 7.383 calls a VBScript function from a JScript new operator to create a Visual Basic safe array. The array itself writes the contents of this two-dimensional array to

the user's page. An alert box is also invoked that contains the value in the second column of the second row.

Listing 7.383 Using the getItem() Method

```
<script type="text/vbscript">
' Define the VB Array
Function myVBAArray()

    ' Define variables for 2-D array positioning
    Dim i
    Dim j

    ' Define variable to hold incremented values to put into
    ' array and assign it an initial value of 1
    Dim k
    k = 1
    ' Create a 2-D array
    Dim myArray(1, 1)

    ' Iterate through 2-D array and put incremented value in
    For i = 0 To 1
        For j = 0 To 1
            myArray(j, i) = k

            ' Write the value to the screen
            document.writeln(k)
            k = k + 1
        Next
        document.writeln("<br>")
    Next

    ' Return the array to the calling function
    myVBAArray = myArray
End Function

</script>
<script type="text/jscript">
<!--

// Create a new instance of VBAArray.
var myArray = new VBAArray(myVBAArray());
alert(myArray.getItem(1,1));

//-->
</script>
```


VBAArray.lbound()

JScript 3.0+

Syntax

```
vbararray.lbound(dimension)  
vbararray.lbound()
```

Description

The `lbound()` method of an instance of a `VBAArray` returns the lowest index value in the *dimension* passed. If no *dimension* is passed, the method defaults to using 1.

Example

Listing 7.384 calls a VBScript function from a JScript new operator to create a Visual Basic safe array. The array itself writes the contents of this two-dimensional array to the user's page. An alert box is also invoked that contains the lowest index number used in the second dimension.

Listing 7.384 Using the `lbound()` Method

```
<script type="text/vbscript">  
  
' Define the VB Array  
Function myVBAArray()  
  
    ' Define variables for 2-D array positioning  
    Dim i  
    Dim j  
  
    ' Define variable to hold incremented values to put into  
    ' array and assign it an initial value of 1  
    Dim k  
    k = 1  
    ' Create a 2-D array  
    Dim myArray(1, 1)  
    ' Iterate through 2-D array and put incremented value in  
    For i = 0 To 1  
        For j = 0 To 1  
            myArray(j, i) = k  
  
            ' Write the value to the screen  
            document.writeln(k)  
            k = k + 1  
        Next  
        document.writeln("<br>")  
    Next  
End Function
```

Listing 7.384 Continued

```
' Return the array to the calling function
myVBAArray = myArray
End Function
```

```
</script>
<script type="text/jscript">
<!--
```

```
// Create a new instance of VBAArray.
var myArray = new VBAArray(myVBAArray());
alert(myArray.lbound(2));
```

```
//-->
</script>
```

VBAArray.toArray()

JScript 3.0+

Syntax

```
vbarray.toArray()
```

Description

The `toArray()` method of an instance of a `VBAArray` returns a valid JScript array from a `VBAArray`.

Example

Listing 7.385 calls a VBScript function from a JScript `new` operator to create a Visual Basic safe array. The array itself writes the contents of this two-dimensional array to the user's page. The array is then converted into a valid JScript array and an alert box is invoked to display a value in this array.

Listing 7.385 Using the `toArray()` Method

```
<script type="text/vbscript">
```

```
' Define the VB Array
Function myVBAArray()

' Define variables for 2-D array positioning
Dim i
Dim j

' Define variable to hold incremented values to put into
' array and assign it an initial value of 1
Dim k
k = 1
```

```

' Create a 2-D array
Dim myArray(1, 1)
' Iterate through 2-D array and put incremented value in
For i = 0 To 1
    For j = 0 To 1
        myArray(j, i) = k

        ' Write the value to the screen
        document.writeln(k)
        k = k + 1
    Next
    document.writeln("<br>")
Next

' Return the array to the calling function
myVBAArray = myArray
End Function

</script>
<script type="text/jscript">
<!--

// Create a new instance of VBAArray.
var myArray = new VBAArray(myVBAArray());

// Convert the VBAArray to a JScript Array.
var myJSArray = myArray.toArray();

// Display the second column, first row value
alert(myJSArray[0,1]);

//-->
</script>

```

VBAArray.ubound()

JScript 3.0+

Syntax

```
vbarrray.ubound(dimension)
```

Description

The `ubound()` method of an instance of a `VBAArray` returns the highest index value in the *dimension* passed.

Example

Listing 7.386 calls a VBScript function from a JScript new operator to create a Visual Basic safe array. The array itself writes the contents of this two-dimensional array to

the user's page. An alert box is also invoked that contains the highest index number used in the second dimension.

Listing 7.386 Using the `ubound()` Method

```
<script type="text/vbscript">

' Define the VB Array
Function myVBAArray()

    ' Define variables for 2-D array positioning
    Dim i
    Dim j

    ' Define variable to hold incremented values to put into
    ' array and assign it an initial value of 1
    Dim k
    k = 1
    ' Create a 2-D array
    Dim myArray(1, 1)
    ' Iterate through 2-D array and put incremented value in
    For i = 0 To 1
        For j = 0 To 1
            myArray(j, i) = k

            ' Write the value to the screen
            document.writeln(k)
            k = k + 1
        Next
        document.writeln("<br>")
    Next

    ' Return the array to the calling function
    myVBAArray = myArray
End Function

</script>
<script type="text/jscript">
<!--

// Create a new instance of VBAArray.
var myArray = new VBAArray(myVBAArray());
alert(myArray.ubound(2));

//-->
</script>
```

void

JavaScript 1.1+, JScript 1.0+, ECMAScript 1E+
Nav3+, NES2+, IE3+

Syntax

```
void(expression)  
void expression
```

Description

The void operator is used to evaluate an expression without returning a value. This operator returns undefined and is commonly used in place of the onClick event handler because of various bugs in early Unix Navigator browsers. Note that Opera browsers do not support this operator.

Example

In Listing 7.387, the void operator is used to keep an <a> link on a page from clicking through anywhere while still invoking the function specified.

Listing 7.387 Using the void Operator to Call a Function

```
<html>  
<head>  
  <title>Examples of the void Operator</title>  
<script type="text/javascript">  
  <!--  
  
  // Define a function to be called by clicking the link.  
  function myFunc(){  
    alert("You clicked the link!");  
  }  
  
  <!-->  
</script>  
</head>  
<body>  
<a HREF="javascript:void(myFunc())">Click here to call the function</A>  
</body>  
</html>
```

volatile

JavaScript 1.2+
Nav4+, NES3+

Syntax

Reserved Keyword

Description

The `volatile` keyword has not been implemented in JavaScript to date. It has been reserved for future use.

Example

This keyword has not been implemented; therefore, no example is provided.

while

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
while(condition){  
    code;  
}
```

Description

The `while` conditional statement evaluates the *condition* passed and executes the code within the block until the condition is no longer met. This is often used to evaluate the value of variables, and then perform tasks as well as to iterate through lines in a file when implemented on the server-side.

Example

Listing 7.388 defines a number and then asks the user for a second number. The user is continually asked for the second number until a number lower than the defined number is entered.

Listing 7.388 *Using the while Statement to Check the Value Passed In*

```
<html>  
<head>  
    <title>Using while</title>  
</head>  
<body>  
<script type="text/javascript">  
!- -  
  
// Define the number to compare against and ask the user for a guess.  
var indexNum = 30;  
var guess = parseInt(prompt("Please enter a number.", ""));  
  
// As long as the user puts in a higher number, keep prompting.  
while(guess >= indexNum){  
    guess = parseInt(prompt("Try again. Guess lower!", ""));  
}
```

```
// Once the user guesses a number lower than the indexed number
// write the following to the screen.
document.write('You have guessed a number lower than the index number. ');
document.write('You guessed ' + guess + ' and the index was ');
document.write(indexNum + '.');

// Close the stream to the browser.
document.close();

//-->
</script>
</body>
</html>
```

with

JavaScript 1.0+, JScript 1.0+, ECMAScript 1E+
Nav2+, NES2+, IE3+, Opera3+

Syntax

```
with(object){
    code;
}
```

Description

The *with* statement takes an *object* and refers to all the properties, methods, and events of that object within the code without directly referencing the object itself. This allows you to use these characteristics of an object, such as the *Math* object, without specifically referencing the object.

Example

Listing 7.389 creates a variable, and then assigns it and the methods called on it, as a *Math* object. Because the *with* statement is used, you do not have to reference the methods via the instance of this object.

Listing 7.389 *Using the with Statement*

```
<script type="text/javascript">
<!--

// Define the variables we are going to use.
var myNum = 25;
var myE, mySin, mySqrt;

// Use a with statement to use the methods of the Math object.
with (Math) {
```

Listing 7.389 Continued

```
    myE = E;  
    mySqrt = sqrt(myNum)  
    mySin = sin(PI/myNum)  
}  
  
//...>  
</script>
```


CHAPTER 8

Client-Side

This chapter contains all the items and elements making up the JavaScript language in browsers. As with the other chapters in this section of the book, each entry includes the version, browser support, syntax, a description, an example, as well as many other details.

JavaScript objects are in alphabetical order to provide you with quick, easy access. The methods, properties, functions, and event handlers of every object appear alphabetically after the respective parent object using the simple dot notation used in all the reference chapters.

NOTE

Throughout this chapter, the following abbreviations will be used to show browser support for each entry. Each abbreviation is followed by the browser version when the entry was first supported and a plus sign (+) if the browser continues to support the entry today:

NES = Netscape Enterprise Server

Nav = Netscape Communicator

IE = Microsoft's Internet Explorer

Opera = Opera

Anchor

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+, Opera5+

Syntax

`document.anchors`

Description

An instance of the Anchor object is created with each instance of the `<a>` tag with the `name` attribute (or `id` attribute) set. An *anchor* itself is a point in an HTML document that can be reached by using a hypertext link. The hypertext link that references an anchor can appear within the same document as the anchor or in a completely different Web page. In either case, anchors allow access to specific parts of Web pages rather than always displaying the top of a document when a Web page is loaded in a browser. For an anchor to act like an anchor, the `name` attribute (or `id` attribute) must be set for hyperlinks to reference the anchor's position in the document.

In HTML, the `<a>` tag is usually accompanied by an `href` attribute that contains a hypertext link. This is the complete opposite of the definition of an anchor because when the `href` is used in conjunction with the `<a>` tag, it is a link rather than an anchor. In fact, JavaScript uses the `Link` object to access the anchor tag when the tag is used as a link.

Table 8.1 lists the properties and methods available from the Anchor object.

Table 8.1 Properties and Methods Used by the Anchor Object

Type	Item	Description
Property	<code>name</code>	A name that provides access to the anchor from a link.
	<code>text</code>	The text that appears between the <code><a></code> and <code></code> tags.
	<code>x</code>	The x coordinate of the anchor.
	<code>y</code>	The y coordinate of the anchor.
Method	<code>unwatch()</code>	This method removes a watch point.
	<code>watch()</code>	This method sets a watch point.

Example

Listing 8.1 demonstrates how anchors are created and used in HTML documents by creating a dictionary of fruit. The heading for each letter of the dictionary is designated as an anchor. At the bottom of the document, some properties of the anchors are accessed using JavaScript.

Listing 8.1 *Creating Anchors in a Document*

```

<html>

<center>
<h1><u>Fruit Dictionary</u></h1>

Pick a letter:
<a href="#A">A</a>|<a href="#B">B</a>|<a href="#C">C</a>
</center>

<br><a name="A"><h4>The Letter A</h4></a>
Apple = A round, juicy fruit that comes in red, yellow, and green.<br>
Apricot = A round yellow-orange fruit.<br>

<br><a name="B"><h4>The Letter B</h4></a>
Banana = A long, yellow, curved fruit with a soft core.<br>
Blackberry = A black, tart fruit from a prickly bush.<br>
Blueberry = A small, round, blue berry that grows on a bush.<br>

<br><a name="C"><h4>The Letter C</h4></a>
Cantaloupe = A large orange melon with hard outer surface.<br>
Coconut = A large round fruit with a hard, fuzzy outer surface.<br>

<script type="text/javascript" language="JavaScript">
<!--
document.write("<br>The Anchor Properties:<br>");

// Display the values of the properties associated with each anchor.
for(var counter=0; counter<document.anchors.length; counter++)
{
    document.write("anchors[" + counter + "].name=");
    document.write(document.anchors[counter].name, "<br>");
    document.write("anchors[" + counter + "].text=");
    document.write(document.anchors[counter].text, "<br>");
    document.write("anchors[" + counter + "].x=");
    document.write(document.anchors[counter].x, "<br>");
    document.write("anchors[" + counter + "].y=");
    document.write(document.anchors[counter].y, "<br>");
}

// -->
</script>

</html>

```

Anchor.name

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+, Opera5+

Syntax

`document.anchors[num].name`

Description

The `name` property holds the name of the anchor and is the way hyperlinks reference the anchor's position. This property is originally set by the `name` attribute in the `<a>` tag.

Example

Listing 8.2 creates a dictionary of fruit where the heading for each letter of the dictionary is designated as an anchor. At the bottom of the document, the `name` property of each anchor is used to create the hyperlink text and reference each anchor.

Listing 8.2 *Accessing the `name` Property of the Anchor Object*

```
<html>

<center>
<h1><u>Fruit Dictionary</u></h1>

Pick a letter:
<a href="#A">A</a>|<a href="#B">B</a>|<a href="#C">C</a>
</center>

<br><a name="A"><h4>The Letter A</h4></a>
Apple = A round, juicy fruit that comes in red, yellow, and green.<br>
Apricot = A round yellow-orange fruit.<br>

<br><a name="B"><h4>The Letter B</h4></a>
Banana = A long, yellow, curved fruit with a soft core.<br>
Blackberry = A black, tart fruit from a prickly bush.<br>
Blueberry = A small, round, blue berry that grows on a bush.<br>

<br><a name="C"><h4>The Letter C</h4></a>
Cantaloupe = A large orange melon with hard outer surface.<br>
Coconut = A large round fruit with a hard, fuzzy outer surface.<br>

<script type="text/javascript" language="JavaScript">
<!--
document.write("<br>Pick a letter:");

// Create a link for each anchor using the Anchor object.
for(var counter=0; counter<document.anchors.length; counter++)
{
```

```

document.write("<a href='#",document.anchors[counter].name,"'>");
document.write(document.anchors[counter].name,"</a>|");
}

// -->
</script>

</html>

```

Anchor.text

JavaScript 1.2-1.4
Nav4-4.5

Syntax

```
document.anchors[num].text
```

Description

The text property contains the text that appears between the <a> and tags. If other HTML tags appear within these two anchor tags, the text property might not contain all the text between the anchor tags.

Example

Listing 8.3 creates a dictionary of fruit where the heading for each letter of the dictionary is designated as an anchor. At the bottom of the document, the text property of the anchors is used to create hyperlink text that points to each anchor.

Listing 8.3 Accessing the text Property of the Anchor Object

```

<html>

<center>
<h1><u>Fruit Dictionary</u></h1>

Pick a letter:
<a href="#A">A</a>|<a href="#B">B</a>|<a href="#C">C</a>
</center>

<br><a name="A"><h4>The Letter A</h4></a>
Apple = A round, juicy fruit that comes in red, yellow, and green.<br>
Apricot = A round yellow-orange fruit.<br>

<br><a name="B"><h4>The Letter B</h4></a>
Banana = A long, yellow, curved fruit with a soft core.<br>
Blackberry = A black, tart fruit from a prickly bush.<br>
Blueberry = A small, round, blue berry that grows on a bush.<br>

```

Listing 8.3 Continued

```
<br><a name="C"><h4>The Letter C</h4></a>
Cantaloupe = A large orange melon with hard outer surface.<br>
Coconut = A large round fruit with a hard, fuzzy outer surface.<br>

<script type="text/javascript" language="JavaScript">
<!--
document.write("<br>Pick a letter:<br>");

// Create a link for each anchor using the Anchor object.
for(var counter=0; counter<document.anchors.length; counter++)
{
    document.write("<a href='#",document.anchors[counter].name,"'>");
    document.write(document.anchors[counter].text,"</a><br>");
}

// -->
</script>

</html>
```

Anchor.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.anchors[num].unwatch(property)
```

Description

The `unwatch()` method of the `Anchor` object is used to turn off the watch for a particular property specified by *property*.

Example

Listing 8.4 shows how the `unwatch()` method is used to stop watching the *name* property of the `Anchor` object after its name has changed to "juice".

Listing 8.4 Example of the `unwatch()` Method of the `Anchor` Object

```
<html>
<a name="oranges">Oranges</a> are orange.<br>
<a name="apples">Apples</a> are red.<br><br>

<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
```

```
document.write(id + " changed from " + oldValue + " to ");
document.write(newValue + "<br>");
return newValue;
}

// Start watch.
document.anchors[0].watch("name", alertme);

document.write("Original anchor name: " + document.anchors[0].name + "<br>");

// Change anchor name associated with oranges.
document.anchors[0].name = "juice";

// End watch.
document.anchors[0].unwatch("name");

document.write("New anchor name: " + document.anchors[0].name + "<br>");

// Change anchor name associated with oranges.
document.anchors[0].name = "orange juice";

document.write("Final anchor name: " + document.anchors[0].name);
// -->
</script>
</html>
```

Anchor.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.anchors[num].watch(property, function)
```

Description

The `watch()` method of the `Anchor` object is used to turn on the watch for a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 8.5 shows how the `watch()` method is used to start watching the *name* property of the `Anchor` object.

Listing 8.5 Example of the `watch()` Method of the Anchor Object

```
<html>
<a name="oranges">Oranges</a> are orange.<br>
<a name="apples">Apples</a> are red.<br><br>

<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch.
document.anchors[0].watch("name",alertme);

document.write("Original anchor name: " + document.anchors[0].name + "<br>");

// Change anchor name associated with oranges.
document.anchors[0].name = "juice";

document.write("New anchor name: " + document.anchors[0].name + "<br>");
// -->
</script>
</html>
```

Anchor.x

JavaScript 1.2-1.4
Nav4-4.5

Syntax

```
document.anchors[num].x
```

Description

The `x` property contains the `x` coordinate of the anchor, in pixels, from the left edge of the document to the anchor.

Example

Listing 8.6 creates a single anchor out of a header at the top of the document. The `x` property is used to display the horizontal position of the Anchor object on the bottom of the page.

Listing 8.6 Accessing the `x` Property of the Anchor Object

```
<html>

<center><a name="A"><h4>Apple</h4></a></center>
```



```
A round, juicy fruit that comes in red, yellow, and green.<br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the x property of the anchor.
document.write("The x property is equal to ",document.anchors[0].x);

// -->
</script>

</html>
```

Anchor.y

JavaScript 1.2-1.4
Nav4-4.5

Syntax

```
document.anchors[num].y
```

Description

The y property contains the y coordinate of the anchor, in pixels, from the top edge of the document to the anchor.

Example

Listing 8.7 creates a single anchor out of a header at the top of the document. The y property is used to display the vertical position of the Anchor object on the bottom of the page.

Listing 8.7 Accessing the y Property of the Anchor Object

```
<html>

<center><a name="B"><h4>Banana</h4></a></center>
A long, yellow, curved fruit with a soft core.<br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the y property of the anchor.
document.write("The y property is equal to ",document.anchors[0].y);

// -->
</script>

</html>
```

Applet

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

Core client-side JavaScript object.

Description

In JavaScript, applets embedded in HTML Web pages are represented by the `Applet` object. This object, which is created with each instance of the `<applet>` tag in a document, allows access to the public fields and methods of the applet through JavaScript properties and methods. Table 8.2 lists the properties and methods used by the `Applet` object.

Table 8.2 *Properties and Methods of the `Applet` Object*

Type	Description
Property	All the public fields in a Java applet are accessed as properties of the <code>Applet</code> object in JavaScript.
Method	All the public methods in a Java applet are accessed as methods of the <code>Applet</code> object in JavaScript.

Example

Assume, for the sake of the example code in Listing 8.8, that you have a calculator applet that you want to embed in an HTML document. Also assume that this applet has a method called `add()`, which adds two numbers and returns the result. If you wanted to access this method from within JavaScript, you would begin by defining the applet using `<applet>` tags. Then the JavaScript code would pass the numbers 2 and 5 into the `add()` method, using dot notation, and the result of 7 would be returned and displayed in the browser.

NOTE

This example will not work unless you have an applet called `calculator` that has a function called `add()`.

Listing 8.8 *Accessing the Method of an `Applet` Object*

```
<html>

<applet name="calculator" code="calculator.class" width=50 height=50></applet>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// Use the calculator applet to add two numbers.
document.write("2+5=",calculator.add(2,5);      // 7 is returned

// -->
</script>
</html>
```

Area

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

An instance of the Area object is created with each occurrence of the <area> tag within an HTML document. In HTML documents, the <area> tag is used in conjunction with the <map> tag to define an area within a picture that will act as a hyperlink. Because the Area object is a hyperlink, it is equivalent to the Link object in JavaScript. In fact, the Area object is stored in the same array where Link objects are stored. Table 8.3 lists the properties, methods, and event handlers of the Area object.

Table 8.3 *Properties, Methods, and Event Handlers Used by the Area Object*

Type	Item	Description
Properties	hash	The portion of the URL that is the anchor, including the # symbol.
	host	The hostname (IP address) and port specified in the URL.
	hostname	The hostname specified within the URL.
	href	The entire URL.
	pathname	The path of the file specified in the URL beginning with the / symbol.
	port	The port specified in the URL.
	protocol	The protocol specified in the URL, including the ending colon (:).
	search	The search part of the URL, including the beginning question mark (?).
Methods	target	The name of the target window in which the URL should be displayed.
	handleEvent()	This method calls the event handler associated with this event.
	unwatch()	This method removes a watch point.
	watch()	This method sets a watch point.

Table 8.3 Continued

Type	Item	Description
Event Handlers	onDb1C1ick	This event handler is invoked when the mouse is double-clicked while in the region defined by the Area object.
	onMouseOut	This event handler is invoked when the mouse moves outside the region defined by the Area object.
	onMouseOver	This event handler is invoked when the mouse moves into the region defined by the Area object.

Example

Listing 8.9 creates a paint store Web page complete with a box of colors that contains hyperlinks to bogus color sites. The hyperlinks are created over the graphic using the <map>, , and <area> tags. When the mouse is moved over an area, the properties associated with that area are displayed in the text fields at the bottom of the screen. Figure 8.1 shows what happens when the mouse pointer is placed in the green box.

Listing 8.9 Creating Areas and Accessing Their Properties

```
<html>
<body>

<h2>The Paint Store</h2>

Select one of the 4 colors to find out more about the colors we carry.<br>

<map name="colorMap">
  <area name="redArea"
    coords="1,1,48,48"
    href="http://www.red.com:1234/red.html?query=red#RED"
    target="_top"
    onMouseOver="overBox(0)"
    onMouseOut="clearBox()" ">
  <area name="greenArea"
    coords="51,1,99,49"
    href="http://www.green.com:5678/green.html?query=green#GREEN"
    target="_top"
    onMouseOver="overBox(1)"
    onMouseOut="clearBox()" ">
  <area name="yellowArea"
    coords="1,51,51,99"
    href="http://www.yellow.com:9876/yellow.html?query=yellow#YELLOW"
    target="_top"
    onMouseOver="overBox(2)"
    onMouseOut="clearBox()" ">
```

```

<area name="blueArea"
      coords="51,51,99,99"
      href="http://www.blue.com:5432/blue.html?query=blue#BLUE"
      target="_top"
      onMouseOver="overBox(3)"
      onMouseOut="clearBox()">
</map>


<br><br><b><u>AREA Properties</u></b>

<form name="myForm">
  hash=<input name="tHash" type="textare"><br>
  host=<input name="tHost" type="textare"><br>
  hostname=<input name="tHostName" type="textare"><br>
  href=<input name="tHref" type="textare"><br>
  pathname=<input name="tPathName" type="textare"><br>
  port=<input name="tPort" type="textare"><br>
  protocol=<input name="tProtocol" type="textare"><br>
  search=<input name="tSearch" type="textare"><br>
  target=<input name="tTarget" type="textare"><br>
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Fill in the text area fields.
function overBox(num)
{
  document.myForm.tHash.value = document.links[num].hash;
  document.myForm.tHost.value = document.links[num].host;
  document.myForm.tHostName.value = document.links[num].hostname;
  document.myForm.tHref.value = document.links[num].href;
  document.myForm.tPathName.value = document.links[num].pathname;
  document.myForm.tPort.value = document.links[num].port;
  document.myForm.tProtocol.value = document.links[num].protocol;
  document.myForm.tSearch.value = document.links[num].search;
  document.myForm.tTarget.value = document.links[num].target;
}

// Clear text in the text area fields.
function clearBox()
{
  document.myForm.tHash.value = "";
  document.myForm.tHost.value = "";
  document.myForm.tHostName.value = "";

```

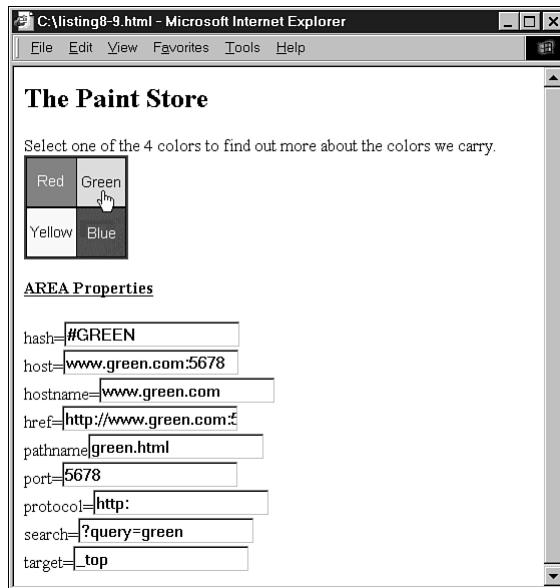
Listing 8.9 Continued

```

document.myForm.tHref.value = "";
document.myForm.tPathName.value = "";
document.myForm.tPort.value = "";
document.myForm.tProtocol.value = "";
document.myForm.tSearch.value = "";
document.myForm.tTarget.value = "";
}
// -->
</script>

</body>
</html>

```

**Figure 8.1***Accessing an area's properties.***Area.handleEvent()**

JavaScript 1.2+
Nav4+

Syntax

```
document.links[num].handleEvent(event)
```

Description

The `handleEvent()` method invokes the event handler, specified by the *event* argument, that is associated with link specified by the *num* argument. The method returns the value associated with the specified *event* argument.

Example

Listing 8.10 creates a Web page for learning more about the toppings used on a pizza. When an area is clicked, the user is taken to a place within the document to learn more about the pizza topping. The `handleEvent()` method is used to pass all the mouse events from the first area (represented by peppers) to the event handlers of the second area (represented by onion). Normally, you would want to display a special notice when the mouse moves over the pepper area, but the `handleEvent()` method is used to make the pepper area do the same thing as the onion area. Notice that "onion" is placed in the text box when the mouse is moved over either of the two defined areas, thanks to the `handleEvent()` method.

Listing 8.10 *Passing Events to Other Area Objects to be Handled*

```
<html>
<body>

<center><h1><u>
Learn more about the toppings we use on our pizza
</u></h1></center>

<h3>Choose a pizza topping from the pictures to learn more.</h3>

<map name="toppingsMap">
  <area name="peppers"
    coords="1,1,48,48"
    href="#PEPPERS"
    target="_top"
    onMouseOver="fillTextField(event)"
    onMouseOut="fillTextField(event)"><br>
  <area name="onion"
    coords="51,1,99,49"
    href="#ONION"
    target="_top"
    onMouseOver="document.pizzaForm.textbox.value='onion'"
    onMouseOut="document.pizzaForm.textbox.value=' '>
</map>

```

Listing 8.10 Continued

```
<br>
<form name="pizzaForm">
  <input type="text"
    name="textbox">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function passes event to another Area object to handle.
function fillTextField(event)
{
  // Pass event to 2nd area link.
  document.links[1].handleEvent(event);
}

// -->
</script>

<a name="PEPPERS"><h3>The peppers we use:</h3></a>
<ul>
  <li>Yellow</li>
  <li>Red</li>
  <li>Green</li>
</ul>

<a name="ONION"><h3>The onions we use:</h3></a>
<ul>
  <li>Mild</li>
  <li>Hot</li>
</ul>

</body>
</html>
```

Area.hash

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.links[num].hash
```

Description

The hash property associated with an Area object contains the anchor specified in the URL including the leading hash symbol (#).

Example

Listing 8.11 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the value of the hash property for the selected link. In this example the value displayed will be one of the following strings: "#SPARKPLUGS", "#TIRES", "#HEADLIGHTS", and "#FUSES". After the alert window is closed, the browser moves to the portion of the document specified by the hash property.

Listing 8.11 *Displaying the hash Property of the Area Object*

```
<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1></center>

<h3>Select a part from the picture.</h3>

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="#SPARKPLUGS"
    target="_top"
    onClick="alert(document.links[0].hash)">
  <area name="tires"
    coords="51,1,99,49"
    href="#TIRES"
    target="_top"
    onClick="alert(document.links[1].hash)">
  <area name="headlights"
    coords="1,51,51,99"
    href="#HEADLIGHTS"
    target="_top"
    onClick="alert(document.links[2].hash)">
  <area name="fuses"
    coords="51,51,99,99"
    href="#FUSES"
    target="_top"
    onClick="alert(document.links[3].hash)">
</map>


<br>
```

Listing 8.11 Continued

```
<a name="SPARKPLUGS"><h3>Spark Plugs</h3></a>
<ul>
  <li>SP93654</li>
  <li>SP34710</li>
  <li>SP19374</li>
</ul>

<a name="TIRES"><h3>Tires</h3></a>
<ul>
  <li>Mud Stompers</li>
  <li>Low Riders</li>
  <li>Standard</li>
</ul>

<a name="HEADLIGHTS"><h3>Headlights</h3></a>
<ul>
  <li>Night Vision bulbs</li>
  <li>Standard</li>
</ul>

<a name="FUSES"><h3>Fuses</h3></a>
<ul>
  <li>Red</li>
  <li>Yellow</li>
  <li>Green</li>
  <li>Blue</li>
</ul>

</body>
</html>
```

Area.host

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.links[num].host`

Description

The `host` property associated with an `Area` object contains the hostname and port that is specified in the URL separated by a colon (:).

Example

Listing 8.12 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the string `"www.wildbillstruckparts.com:4000"`, which

is the value of the host property for the selected link. After the alert window is closed, the browser will attempt to load the specified file located on the fictitious Web site.

Listing 8.12 *Displaying the host Property of the Area Object*

```
<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1>

<h3>Select a part from the picture.</h3>

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="http://www.wildbillstruckparts.com:4000/sparkplugs.html"
    target="_top"
    onClick="alert(document.links[0].host)">
  <area name="tires"
    coords="51,1,99,49"
    href="http://www.wildbillstruckparts.com:4000/tires.html"
    target="_top"
    onClick="alert(document.links[1].host)">
  <area name="headlights"
    coords="1,51,51,99"
    href="http://www.wildbillstruckparts.com:4000/headlights.html"
    target="_top"
    onClick="alert(document.links[2].host)">
  <area name="fuses"
    coords="51,51,99,99"
    href="http://www.wildbillstruckparts.com:4000/fuses.html"
    target="_top"
    onClick="alert(document.links[3].host)">
</map>


</center>

</body>
</html>
```

Area.hostname

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`document.links[num].hostname`

Description

The `hostname` property associated with the `Area` object contains just the hostname that is specified in the URL.

Example

Listing 8.13 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the string "www.wildbillstruckparts.com", which is the value of the `hostname` property for the selected link. After the alert window is closed, the browser will attempt to load the file located on the fictitious Web site.

Listing 8.13 *Displaying the `hostname` Property of the `Area` Object*

```
<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1>

<h3>Select a part from the picture.</h3>

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="http://www.wildbillstruckparts.com:4000/sparkplugs.html"
    target="_top"
    onClick="alert(document.links[0].hostname)">
  <area name="tires"
    coords="51,1,99,49"
    href="http://www.wildbillstruckparts.com:4000/tires.html"
    target="_top"
    onClick="alert(document.links[1].hostname)">
  <area name="headlights"
    coords="1,51,51,99"
    href="http://www.wildbillstruckparts.com:4000/headlights.html"
    target="_top"
    onClick="alert(document.links[2].hostname)">
  <area name="fuses"
    coords="51,51,99,99"
    href="http://www.wildbillstruckparts.com:4000/fuses.html"
    target="_top"
    onClick="alert(document.links[3].hostname)">
```

```

</map>


</center>

</body>
</html>

```

Area.href

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.links[num].href
```

Description

The href property associated with the Area property contains the entire URL specified by the href attribute of the <area> tag. This property is a read/write string.

Example

In Listing 8.14, a sports car picture site is created. Selecting your location determines from which site a large picture of the sports car is retrieved. This is accomplished by modifying the href property to link to a completely different site based on the user's choice of cars.

Listing 8.14 *Modifying the href Property of the Area Object*

```

<html>
<body>

<center><h1><u>Sports Car Pictures</u></h1></center>

<h3>Step 1: Choose a car.</h3>

<form name="question">
<input name="carType"
      type="radio"
      checked
      onClick="updateLinks('LAMBORGHINI')">Lamborghini<br>
<input name="carType"
      type="radio"
      onClick="updateLinks('FERRARI')">Ferrari

```

Listing 8.14 Continued

```
</form>

<h3>Step 2: Click on the picture to see the car.</h3>

<map name="carMap">
  <area name="car"
    coords="1,1,48,48"
    href="http://www.cars.com:5678/cars.html?query=fast#LAMBORGHINI"
    target="_top">
</map>


<br>

<script type="text/javascript" language="JavaScript">
<!--

// update the picture host links based on the radio box settings
function updateLinks(car)
{
  // Set to Lamborghini site
  if(car=="LAMBORGHINI")
    document.links[0].href="http://www.cars.com:5678/cars.html
    ↪?query=fast#LAMBORGHINI";

  // Set to Ferrari site
  if(car=="FERRARI")
    document.links[0].href="http://www.sportscars.com:2020/fastcars.html?
    ↪query=fast#FERRARI";
}

// -->
</script>

</body>
</html>
```

Area.onDbClick

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

onDbClick="command"

Description

The `onDb1Click` event handler associated with the `Area` object is invoked when the user double-clicks the mouse pointer while in the region defined by the `<area>` tag.

Example

Listing 8.15 creates a clothing site with a picture of a shirt. Double-clicking the shirt displays a message about red shirts, thanks to the `onDb1Click` event handler.

Listing 8.15 Using the `onDb1Click` Property of the `Area` Object

```
<html>
<body>

<h2>The Clothes Site</h2>

<map name="clothesMap">
  <area name="redShirt"
        coords="1,1,48,48"
        target="_top"
        onDb1Click="alert('You must like red shirts.')">
</map>


</body>
</html>
```

Area.onMouseOut

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
onMouseOut="command"
```

Description

The `onMouseOut` event handler associated with the `Area` object is invoked when the user moves the mouse pointer into the region defined by the `<area>` tag.

Example

Listing 8.16 creates a clothing site with a picture of a shirt. Clicking the shirt graphic takes you to a site about red shirts. If the mouse is moved out of the region, an alert box posts a message to the screen, thanks to the `onMouseOut` event handler.

Listing 8.16 Using the `onMouseOut` Property of the `Area` Object

```
<html>
<body>
```

Listing 8.16 Continued

```
<h2>The Clothes Site</h2>

<map name="clothesMap">
  <area name="redShirt"
        coords="1,1,48,48"
        href="http://www.clothes.com/redShirts.html"
        target="_top"
        onMouseOut="alert('You did not want to go the red shirt site?')">
</map>


</body>
</html>
```

Area.onMouseOver

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
onMouseOver="command"
```

Description

The `onMouseOver` event handler is invoked when the user moves the mouse pointer over the region defined by the `<area>` tag.

Example

Listing 8.17 creates a clothing site with a picture of a shirt. Clicking the shirt graphic takes you to a site about red shirts. Simply moving the mouse over the region causes an alert box to be posted alerting the user that a mouseover event occurred.

Listing 8.17 Using the `onMouseOver` Property of the `Area` Object

```
<html>
<body>

<h2>The Clothes Page</h2>

Click on the sales tag to see the clothes that are on sale:<br>

<map name="clothesMap">
  <area name="sale"
        coords="1,1,100,100"
        href="http://www.clothes.com/specials.html"
        target="_top"
        onMouseOver="alert('An onMouseOver event occurred.')">
```



```

</map>


</body>
</html>

```

Area.pathname

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.links[num].pathname`

Description

The `pathname` property contains the path of the file specified in the URL, including the leading slash (/).

Example

Listing 8.18 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the value of the `pathname` property for the selected link. After the alert window is closed, the browser attempts to load the specified file.

Listing 8.18 *Displaying the `pathname` Property of an `Area` Object*

```

<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1>

<h3>Select a part from the picture.</h3>

<map name="partsMap">
  <area name="sparkPlug"
        coords="1,1,48,48"
        href="sparkplugs.html"
        target="_top"
        onClick="alert(document.links[0].pathname)">
  <area name="tires"
        coords="51,1,99,49"
        href="tires.html"
        target="_top"
        onClick="alert(document.links[1].pathname)">
  <area name="headlights"
        coords="1,51,51,99"
        href="headlights.html"

```

Listing 8.18 Continued

```
        target="_top"
        onClick="alert(document.links[2].pathname)">
<area name="fuses"
      coords="51,51,99,99"
      href="fuses.html"
      target="_top"
      onClick="alert(document.links[3].pathname)">
</map>


</center>

</body>
</html>
```

Area.port

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

```
document.links[num].port
```

Description

The port property contains just the port specified in the URL.

Example

Listing 8.19 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the string "4000", which is the value of the port property for the selected link. After the alert window is closed, the browser will attempt to load the specified Web site.

Listing 8.19 Displaying the port Property of an Area Object

```
<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1>

<h3>Select a part from the picture.</h3>
```

```

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="http://www.wildbillstruckparts.com:4000/sparkplugs.html"
    target="_top"
    onClick="alert(document.links[0].port)">
  <area name="tires"
    coords="51,1,99,49"
    href="http://www.wildbillstruckparts.com:4000/tires.html"
    target="_top"
    onClick="alert(document.links[1].port)">
  <area name="headlights"
    coords="1,51,51,99"
    href="http://www.wildbillstruckparts.com:4000/headlights.html"
    target="_top"
    onClick="alert(document.links[2].port)">
  <area name="fuses"
    coords="51,51,99,99"
    href="http://www.wildbillstruckparts.com:4000/fuses.html"
    target="_top"
    onClick="alert(document.links[3].port)">
</map>


</center>

</body>
</html>

```

Area.protocol

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`document.links[num].protocol`

Description

The `protocol` property contains the protocol (`http:`, `file:`, `ftp:`, and so on) specified in the URL, including the ending colon (`:`).

Example

In Listing 8.20, the `protocol` property containing `http:` is displayed below the area link.

Listing 8.20 Accessing the protocol *Property of an Area Object*

```
<html>

<h2>The Color Page</h2>

<map name="colorMap">
  <area name="greenArea"
    coords="1,1,48,48"
    href="http://www.green.com:5678/green.html?query=green#GREEN"
    target="_top">
</map>


<script>
<!--

// Display the protocol associated with the area
document.write("protocol = ",document.links[0].protocol);

// -->
</script>
</html>
```

Area.search

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

```
document.links[num].search
```

Description

The search property contains the search string specified in the URL, including the leading question mark (?).

Example

Listing 8.21 creates a truck parts page that allows users to find model numbers of parts by clicking a picture of the part. When the user selects a part, she is presented with an alert window that contains the value of the search property for the selected link. In this example, the value displayed will be one of the following strings: "?sparkplugs", "?tires", "?headlights", or "?fuses". After the alert window is closed, the browser attempts to load the fictitious page.

Listing 8.21 Displaying the search *Property of an Area Object*

```
<html>
<body>
```

```

<center><h1><u>Wild Bill's Truck Parts</u></h1>

<h3>Select a part from the picture.</h3>

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="parts.html?sparkplugs"
    target="_top"
    onClick="alert(document.links[0].search)">
  <area name="tires"
    coords="51,1,99,49"
    href="parts.html?tires.html"
    target="_top"
    onClick="alert(document.links[1].search)">
  <area name="headlights"
    coords="1,51,51,99"
    href="parts.html?headlights.html"
    target="_top"
    onClick="alert(document.links[2].search)">
  <area name="fuses"
    coords="51,51,99,99"
    href="parts.html?fuses.html"
    target="_top"
    onClick="alert(document.links[3].search)">
</map>


</center>

</body>
</html>

```

Area.target

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.links[num].target`

Description

The `target` property contains the name of the target window or frame in which the URL should be displayed. This property is a read/write string.

Example

In Listing 8.22, a truck parts site is created that allows the user to find out what parts the store carries. The target of the spark plugs and tires link is modified to make the information appear in a new browser window. Figure 8.2 shows the result of selecting the sparkplug image.

Listing 8.22 *Modifying the target Property of an Area Object*

```
<html>
<body>

<center><h1><u>Wild Bill's Truck Parts</u></h1></center>

<h3>Choose Part from Pictures.</h3>

<map name="partsMap">
  <area name="sparkPlug"
    coords="1,1,48,48"
    href="#SPARKPLUGS"
    target="_top">
  <area name="tires"
    coords="51,1,99,49"
    href="#TIRES"
    target="_top">
  <area name="headlights"
    coords="1,51,51,99"
    href="#HEADLIGHTS"
    target="_top">
  <area name="fuses"
    coords="51,51,99,99"
    href="#FUSES"
    target="_top">
</map>


<br>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// Make the result of the sparkplugs and tires appear in a new window.
document.links[0].target="_blank";    // sparkplugs
document.links[1].target="_blank";    // tires

// -->
</script>

<a name="SPARKPLUGS"><h3>Truck Spark Plugs</h3></a>
<ul>
  <li>SP93654</li>
  <li>SP34710</li>
  <li>SP19374</li>
</ul>

<a name="TIRES"><h3>Truck Tires</h3></a>
<ul>
  <li>Mud Stompers</li>
  <li>Low Riders</li>
  <li>Standard</li>
</ul>

<a name="HEADLIGHTS"><h3>Truck Headlights</h3></a>
<ul>
  <li>Night Vision bulbs</li>
  <li>Standard</li>
</ul>

<a name="FUSES"><h3>Truck Fuses</h3></a>
<ul>
  <li>Red</li>
  <li>Yellow</li>
  <li>Green</li>
  <li>Blue</li>
</ul>

</body>
</html>
```

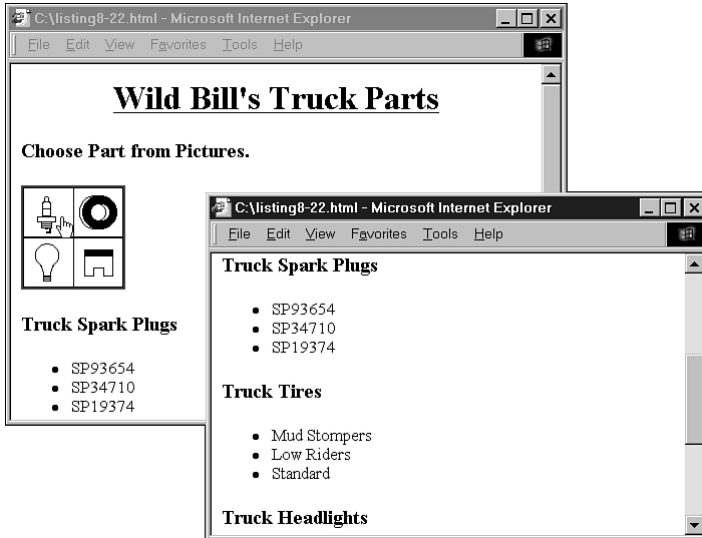


Figure 8.2

A new window opens as a result of clicking the spark plug image.

Area.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.links[num]. unwatch(property)
```

Description

The `unwatch()` method of the `Area` object is used to turn off the watch for a particular property specified by *property*.

Example

Listing 8.23 shows how the `unwatch()` method is used to stop watching the *href* property of the `Area` object after its name has changed to "http://www.toys.com".

Listing 8.23 Example of the `unwatch()` Method of the `Area` Object

```
<html>
```

```
<h2>The Department Store</h2>
```

```
Select a department<br>
```



```

<map name="storeMap">
  <area name="toyArea"
    coords="1,1,100,50"
    href="toys.html"
    target="_top"
  <area name="clothesArea"
    coords="1,50,100,100"
    href="clothes.html"
    target="_top"
</map>

<br><br>

<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
  document.write(id + " changed from " + oldValue + " to ");
  document.write(newValue + "<br>");
  return newValue;
}

// Start watch
document.links[0].watch("href",alertme);

document.write("Original toy department href: ");
document.write(document.links[0].href + "<br>");

// Change href
document.links[0].href = "http://www.toys.com";

document.write("New toy department href: " + document.links[0].href + "<br>");

// Stop watch
document.links[0].unwatch("href",alertme);

// Change href again
document.links[0].href = "http://www.toysrus.com";

document.write("Final toy department href: ");
document.write(document.links[0].href + "<br>");
// -->
</script>
</html>

```

Area.watch()

JavaScript 1.2+

Nav4+, NES3+

Syntax

```
document.links[num]. watch(property, function)
```

Description

The `watch()` method of the `Area` object is used to turn on the watch for a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 8.24 shows how the `watch()` method is used to start watching the *href* property of the `Area` object.

Listing 8.24 Example of `watch()` Method of the `Area` Object

```
<html>

<h2>The Department Store</h2>

Select a department<br>

<map name="storeMap">
  <area name="toyArea"
    coords="1,1,100,50"
    href="toys.html"
    target="_top"
  <area name="clothesArea"
    coords="1,50,100,100"
    href="clothes.html"
    target="_top"
  </map>

<br><br>

<script type="text/javascript" language="JavaScript">
<!--
function alertme(id,oldValue,newValue)
{
  document.write(id + " changed from " + oldValue + " to ");
  document.write(newValue + "<br>");
  return newValue;
}
```

```
// Start watch
document.links[0].watch("href", alertme);

document.write("Original toy department href: ");
document.write(document.links[0].href + "<br>");

// Change href
document.links[0].href = "http://www.toys.com";

document.write("New toy department href: " + document.links[0].href + "<br>");

// -->
</script>
</html>
```

Button

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Button object represents a graphical button that the user can click to initiate an action. Buttons are created as part of a form by using the `<input>` tag with the `type` attribute set to "button" in an HTML document. Once created, buttons can be accessed in JavaScript as an element of a form using dot notation. The arguments, properties, methods, and event handlers for the Button object are listed in Table 8.4.

Table 8.4 Arguments, Properties, Methods, and Event Handlers Associated with the Button Object

Type	Item	Description
Argument	string	The string to appear in the graphical representation of a button.
	num	An index number that allows access to buttons through a form's elements list.
Property	form	This property returns the form object of a button.
	name	The string that is specified in the <code>name</code> attribute of the HTML <code><input></code> tag.
	type	The string that is specified in the <code>type</code> attribute of the HTML <code><input></code> tag. This string is always "button" for the Button object.

Table 8.4 Continued

Type	Item	Description
Method	value	The string that appears in the graphical representation of a button.
	blur()	This method removes focus from a button.
	click()	This method calls the button's <code>onClick</code> event handler.
	focus()	This method applies focus to a button.
	handleEvent()	This method passes an event to the appropriate event handler associated with a button.
Event Handler	unwatch()	This method removes a watch point.
	watch()	This method sets a watch point.
	onBlur	The handler invoked when focus is removed from a button.
	onClick	The handler invoked when the left mouse button is clicked.
	onFocus	The handler invoked when focus is applied to a button.
	onMouseDown	The handler invoked when the left mouse button is clicked to select a button.
	onMouseUp	The handler invoked when the left mouse button is released from clicking a button.

Example

In Listing 8.25, a button is created by using the `<input>` tag. When the button is clicked, the button's name is displayed in the adjacent text box.

Listing 8.25 Creating a Button and Displaying Its Name

```
<html>

<h2>The Button's NAME Property</h2>

<form name="myForm">
  <input type="button"
    value="Press here to see the name of this button"
    name="myBigButton"
    onClick="displayButtonName()">
  <input type="text"
    name="textBox">
</form>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// This function displays the button's name in the textbox.
function displayButtonName()
{
    // Display button name in textbox.
    document.myForm.textBox.value=document.myForm.myBigButton.name;
}

// -->
</script>

</html>
```

Button.blur()

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

```
document.form.button.blur()
```

Description

The `blur()` method removes the focus from a button.

CAUTION

In the Unix versions of Navigator 2 and Navigator 3, the `blur()` method does not work for buttons.

Example

In Listing 8.26, two buttons are created by using the `<input>` tag to demonstrate focus. The first button retains focus after being clicked, but the second button loses focus as soon as it is clicked due to the use of the `blur()` method. There are not a lot of uses for this method, but it is provided for your use all the same.

Listing 8.26 Removing Focus from a Button with the `blur()` Method

```
<html>
```

```
<h2>The Button Focus Game</h2>
```

Click both buttons. Notice that the second button does not hold its focus after being clicked.

```
<form name="myForm">
  <input type="button"
    value="I hold my focus after a click"
    name="button1"><br>
```

Listing 8.26 Continued

```
<input type="button"
      value="I cannot hold my focus after a click"
      name="button2"
      onClick="removeFocus()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function takes the focus off of button2.
function removeFocus()
{
    // Remove the focus from button2.
    document.myForm.button2.blur();
}

// -->
</script>

</html>
```

Button.click()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

```
document.form.button. click()
```

Description

The `click()` method simulates the `click` event.

Example

In Listing 8.27, two buttons are created by using the `<input>` tag. The first button displays an alert box when it is clicked. When the second button is clicked, it causes the first button's `onClick` event handler to be activated, displaying the alert box associated with the first button.

Listing 8.27 Simulating a click Event with the `click()` Method

```
<html>

<form name="myForm">
  <input type="button"
        value="Display alert box"
        name="button1"
        onClick="alert('You clicked the first button.')"><br>
```

```
<input type="button"
      value="Call on button 1"
      name="button2"
      onClick="clickFirstButton()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function activates the first button's onClick handler.
function clickFirstButton()
{
    // Click first button
    document.myForm.button1.click();
}

// -->
</script>

</html>
```

Button.focus()

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`document.form.button.focus()`

Description

The `focus()` method applies focus to the button without invoking the button's `onFocus` event handler.

CAUTION

In the Unix versions of Navigator 2 and Navigator 3, the `focus()` method does not work for buttons.

Example

In Listing 8.28, two buttons are created to demonstrate focus. Choosing the second button causes the focus to shift to the first button instead of the focus staying on the second button, thanks to the `focus()` method.

Listing 8.28 *Shifting the Focus to a Button Using the `focus()` Method*

```
<html>

<h2>The Button Focus Game</h2>

Click both buttons. Notice that when the second button is clicked
focus is shifted to the first button.

<form name="myForm">
  <input type="button"
    value="I hold my focus after a click"
    name="button1"><br>
  <input type="button"
    value="I shift my focus when clicked"
    name="button2"
    onClick="moveFocus()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function puts button1 in focus.
function moveFocus()
{
  // Give button1 the focus.
  document.myForm.button1.focus();
}

// -->
</script>

</html>
```

Button.form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.form.button.form`

Description

The `form` property provides access to the button's parent `Form` object.

Example

Listing 8.29 proves that the button's `form` property contains the parent form object by evaluating the `if` statement to `true`. The sentence "myButton's `form` property is equal to myForm object" is displayed in the browser.

Listing 8.29 Accessing a Button's Parent with the `form` Property

```

<html>
<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Does the parent of the myButton equal myForm?
if(document.myForm.myButton.form == document.myForm)
  alert("myButton's form property is equal to myForm object");    // Equal
else
  alert("myButton's form property is NOT equal to myForm object"); // Not equal

// -->
</script>

</html>

```

Button.handleEvent()

JavaScript 1.2-1.3
Nav4-4.5

Syntax

```
document.form.button.handleEvent(event)
```

Description

The `handleEvent()` method provides a way to invoke a button's event handler, even though the event was not triggered by the user. The *event* argument specifies the Event object associated with the event handler to be invoked.

Example

The code in Listing 8.30 provides information about a car that is for sale. By choosing one of the buttons labeled Doors, Engine, or Transmission, the related information is entered into the text boxes. The text boxes can be cleared by clicking the Clear All Info button. Unlike the other buttons, the Show All Info button passes the `click` event along to each of the other buttons using the `handleEvent()` method. This way, each button takes care of filling in its respective text box.

Listing 8.30 Handling the `click` Event with the `handleEvent()` Method

```

<html>

<center><h2>Car For Sale!</h2></center>

```

Listing 8.30 Continued

Click on a button for more information:

```
<form name="myForm">
  <input type="button"
    value="Doors"
    name="doorsButton"
    onClick="document.myForm.doorsBox.value='4 doors'">
  <input type="text"
    name="doorsBox"><br>
  <input type="button"
    value="Engine"
    name="engineButton"
    onClick="document.myForm.engineBox.value='4 cylinder'">
  <input type="text"
    name="engineBox"><br>
  <input type="button"
    value="Transmission"
    name="transmissionButton"
    onClick="document.myForm.transmissionBox.value='manual transmission'">
  <input type="text"
    name="transmissionBox"><br>
  <input type="button"
    value="Clear All Info"
    name="clearAllButton"
    onClick="clearAll()">
  <input type="button"
    value="Show All Info"
    name="showAllButton"
    onClick="showAll(event)">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function clears all the text boxes.
function clearAll()
{
  document.myForm.doorsBox.value="";
  document.myForm.engineBox.value="";
  document.myForm.transmissionBox.value="";
}

// This function passes the click event to all of the buttons.
// Each button then fills its respective information into the text boxes.
function showAll(event)
{
```

```

    document.myForm.doorsButton.handleEvent(event);
    document.myForm.engineButton.handleEvent(event);
    document.myForm.transmissionButton.handleEvent(event);
}

// -->
</script>

</html>

```

Button.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.name.button.name`

Description

The name property provides access to the name attribute of the button as specified by the `<input>` tag. This property is a read-only string.

Example

In Listing 8.31, a button is created. When the button is clicked, the button's name property is displayed.

Listing 8.31 Accessing a Button's Name with the `name` Property

```

<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onClick="displayMessage()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function displays an alert box the contains the name of the button.
function displayMessage()
{
  // Create a string with name of button.
  var alertString = String("You pressed the button named ");
  alertString += document.myForm.myButton.name;

```

Listing 8.31 Continued

```
// Create alert box.
alert(alertString);
}

// -->
</script>

</html>
```

Button.onBlur

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`onBlur="command"`

Description

The `onBlur` event handler is defined in an `<input>` tag and specifies what to do when the button loses focus.

Example

The code in Listing 8.32 uses the `onBlur` event handler to display an alert box any time the button loses focus.

Listing 8.32 Use the `onBlur` Event Handler to Display an Alert Box When the Button Loses Focus

```
<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onBlur="alert('Blur event occurred')">
</form>

</html>
```

Button.onClick

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera5+

Syntax

`onClick="command"`

Description

The `onClick` event handler is defined in an `<input>` tag and specifies what to do when the button is clicked.

Example

The code in Listing 8.33 uses the `onClick` event handler to display an alert box any time the button is clicked.

Listing 8.33 Use the `onClick` Event Handler to Display an Alert Box When the Button Is Clicked

```
<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onClick="alert('Click event occurred')">
</form>

</html>
```

Button.onFocus

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`onFocus="command"`

Description

The `onFocus` event handler is defined in an `<input>` tag and specifies what to do when the button gains focus.

Example

The code in Listing 8.34 uses the `onFocus` event handler to display an alert box any time the button gains focus.

Listing 8.34 Use the `onFocus` Event Handler to Display an Alert Box When the Button Gains Focus

```
<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onFocus="alert('Focus event occurred')">
</form>
```

Listing 8.34 Continued

```
</form>

</html>
```

Button.onMouseDown

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

```
onMouseDown="command"
```

Description

The onMouseDown event handler is defined in an <input> tag and specifies what to do when the left mouse button is clicked.

Example

The code in Listing 8.35 uses the onMouseDown event handler to display an alert box any time the button is clicked.

Listing 8.35 Use the onMouseDown Event Handler to Display an Alert Box When the Button Is Clicked

```
<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onMouseDown="alert('MouseDown event occurred')">
</form>

</html>
```

Button.onMouseUp

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

```
onMouseUp="command"
```

Description

The onMouseUp event handler is defined in an <input> tag and specifies what to do when the left mouse button is released while over the button.

Example

The code in Listing 8.36 uses the `onMouseUp` event handler to display an alert box any time the left mouse button is released while over the button.

Listing 8.36 Use the `onMouseUp` Event Handler to Display an Alert Box When the Mouse Button Is Released over the Button

```
<html>

<form name="myForm">
  <input type="button"
    value="Big Button"
    name="myButton"
    onMouseUp="alert('MouseUp event occurred')">
</form>

</html>
```

Button.type

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.form.button.type`

Description

The `type` property provides access to the `type` attribute of the button. This property is a read-only string that always contains "button".

Example

Listing 8.37 creates buttons that relate to various math problems. The `type` property is used to determine the number of buttons on the page. This number is then used in the instructions that are displayed on the bottom of the page.

Listing 8.37 Accessing a Button's `type` Property

```
<html>

<h2>The Math Page</h2>

<form name="mathForm">
  <input type="button"
    name="4plus2"
    value="(4 + 2)"
    onClick="document.mathForm.answer.value='(4 + 2) = 6'">
  <input type="button"
    name="4minus2"
```

Listing 8.37 Continued

```
        value="(4 - 2)"
        onClick="document.mathForm.answer.value='(4 - 2) = 2' "><hr>
    Answer:
    <input type="text"
        name="answer">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Create a counter to count the number of buttons in mathForm.
var buttonCount = 0;

// Loop through all the elements of mathForm.
for(var x=0; x<document.mathForm.length; x++)
{
    // Is element a button?
    if(document.mathForm.elements[x].type=="button")
        buttonCount++;    // Increment buttonCount
}

// Display the number of buttons in the mathForm.
document.write("Please select one of the ",buttonCount);    // 2 buttons
document.write(" buttons above to find out the answer to the math problem.");

// -->
</script>

</html>
```

Button.unwatch()

JavaScript 1.2
Nav4, NES3+

Syntax

```
document.form.button.unwatch(property)
```

Description

The `unwatch()` method of the `Button` object is used to turn off the watch for a particular property specified by *property*.

Example

Listing 8.38 shows how the `unwatch()` method is used to stop watching the *value* property of the `Button` object after its name has changed to "Blue".

CAUTION

A bug associated with the `watch()` and `unwatch()` methods prevents the actual text in the button from getting redrawn even though the button's *value* property does get changed.

Listing 8.38 Example of the `unwatch()` Method of the `Button` Object

```
<html>

<form name="colorForm">
<input type="button" name="colorButton" value="Red">
</form>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.colorForm.colorButton.watch("value",alertme);

document.write("Original button value: ");
document.write(document.colorForm.colorButton.value + "<br>");

// Change button text
document.colorForm.colorButton.value = "Blue";

document.write("New button value: ");
document.write(document.colorForm.colorButton.value + "<br>");

// Stop watch
document.colorForm.colorButton.unwatch("value");

// Change button text
document.colorForm.colorButton.value = "Green";

document.write("Final button value: ");
document.write(document.colorForm.colorButton.value + "<br>");

// -->
</script>
</html>
```

Button.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.form.button.value`

Description

The `value` property provides access to the `value` attribute of the button. This property is a read-only string that is displayed in the graphical button.

Example

Listing 8.39 uses the `value` property to customize the text in a button. To modify the button value, enter your name into the text field and then click the Customize the Button button. The button at the bottom of the document will no longer contain just the word "Press".

Listing 8.39 *Modifying Text in a Button Using the `value` Property*

```
<html>

<h2>Customize the Button</h2>

<form name="myForm">
  Please enter your name:
  <input type="text"
    name="nameBox">
  <input type="button"
    name="customizeTheButton"
    value="Customize the button"
    onClick="customizeButton()"><br>
  <input type="button"
    name="theButton"
    value="Press"
    onClick="displayAlert()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function customizes the button.
function customizeButton()
{
  // Create a string using the name entered in text box.
  var aString = document.myForm.nameBox.value;
  aString += " press this button!";
```

```

    // Change the value displayed in the button.
    document.myForm.theButton.value=aString;
}

// This function creates an alert box.
function displayAlert()
{
    // Create an alert box.
    alert("You pressed the button.");
}

// -->
</script>

</html>

```

Button.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.form.button.watch(property, function)
```

Description

The `watch()` method of the `Button` object is used to turn on the watch for a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 8.40 shows how the `watch()` method is used to start watching the *value* property of the `Button` object.

CAUTION

A bug associated with the `watch()` and `unwatch()` methods prevents the actual text in the button from getting redrawn even though the button's *value* property does get changed.

Listing 8.40 Example of the `watch()` Method of the `Button` Object

```

<html>

<form name="colorForm">
<input type="button" name="colorButton" value="Red">
</form>

```

Listing 8.40 Continued

```
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.colorForm.colorButton.watch("value",alertme);

document.write("Original button value: ");
document.write(document.colorForm.colorButton.value + "<br>");

// Change button text
document.colorForm.colorButton.value = "Blue";

document.write("New button value: ");
document.write(document.colorForm.colorButton.value + "<br>");

// -->
</script>
</html>
```

Checkbox

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Checkbox object represents a graphical check box that the user can click to toggle the check on or off. Check boxes are created as part of a form by using the `<input>` tag with the `type` attribute set to `checkbox` in an HTML document. Once created, check boxes can be accessed in JavaScript as an element of a form using dot notation. Check boxes can also be grouped together under the same name and accessed as an array by using brackets. The arguments, properties, methods, and event handlers of the Checkbox object are listed in Table 8.5

Table 8.5 Arguments, Properties, Methods, and Event Handlers Associated with the `Checkbox` Object

Type	Item	Description
Argument	num1	An index number that allows access to check boxes through a form's element list.
	num2	An index number that allows access to individual check boxes that are grouped together under the same name.
Property	checked	A boolean value that determines whether the check box is checked.
	defaultChecked	A boolean value that holds the initial state of the check box. This value is set with the <code>checked</code> attribute.
	form	This property returns the <code>Form</code> object of the check box.
	name	The string that is specified in the <code>name</code> attribute of the HTML <code><input></code> tag.
	type	The string that is specified in the <code>type</code> attribute of the HTML <code><input></code> tag. This string is always <code>"checkbox"</code> for the <code>Checkbox</code> object.
	value	The value returned when the form is submitted.
Method	blur()	This method removes focus from the check box.
	click()	This method calls the check box's <code>onClick</code> event handler.
	focus()	This method applies focus to this check box.
	handleEvent()	This method passes an event to the appropriate event handler associated with the check box.
	unwatch() watch()	This method removes a watch point. This method sets a watch point.
Event Handler	onBlur	The handler invoked when focus is removed from the check box.
	onClick	The handler invoked when the check box is selected.
	onFocus	The handler invoked when focus is applied to the check box.

Example

Listing 8.41 creates a hamburger order page. Check boxes are used to select toppings for the hamburger. When the Submit Order button is clicked, an alert box is displayed with the selected toppings.

Listing 8.41 *Creating Check Boxes and Accessing Some of Their Properties*

```
<html>

<h2>Hamburger Order</h2>

Step 1: Please select the toppings you would like on your hamburger:<br>

<form name="orderForm">
  Lettuce
  <input type="checkbox"
    value="lettuce"
    name="lettuceCB"><br>
  Cheese
  <input type="checkbox"
    value="cheese"
    name="cheeseCB"><br>
  Tomato
  <input type="checkbox"
    value="tomato"
    name="tomatoCB"><br>
  Step 2:
  <input type="button"
    value="Submit Order"
    name="orderButton"
    onClick="submitOrder()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function creates an alert box to display which toppings were selected.
function submitOrder()
{
  // Create a string to display in the alert box
  var alertString = String("Order: Hamburger ");
  if(document.orderForm.lettuceCB.checked == true)
    alertString += " with lettuce ";      // Add lettuce to string
  if(document.orderForm.cheeseCB.checked == true)
    alertString += "with cheese ";      // Add cheese to string
  if(document.orderForm.tomatoCB.checked == true)
    alertString += "with tomato ";      // Add tomato to string

  // Create alert box
  alert(alertString);
}
```

```
// -->
</script>

</html>
```

Checkbox.blur()

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

```
document.form.checkbox.blur()
```

Description

The `blur()` method removes the focus from a check box.

CAUTION

In the Unix versions of Navigator 2 and Navigator 3, the `blur()` method does not work for check boxes.

Example

In Listing 8.42, two check boxes are created by using the `<input>` tag. The first check box retains focus after being clicked, but the second check box loses focus as soon as it is clicked because of the use of the `blur()` method. Do not be surprised if you do not see a difference between the check boxes in this example. In some browsers, you will not be able to detect the difference between a check box that is focused versus one that is not focused. For this reason, you will probably find very little use for this method.

Listing 8.42 Removing Focus from a Check Box with the `blur()` Method

```
<html>
```

```
<h2>The Checkbox Focus Test</h2>
```

Click both check boxes. Notice that the second check box does not hold its focus after being clicked.

```
<form name="myForm">
  I hold my focus after a click
  <input type="checkbox"
    name="checkBox1"><br>
  I cannot hold my focus after a click
  <input type="checkbox"
    name="checkBox2"
    onClick="removeFocus()">
```

Listing 8.42 Continued

```
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function takes the focus off of checkBox2.
function removeFocus()
{
    // Remove the focus from checkBox2
    document.myForm.checkBox2.blur();
}

// -->
</script>

</html>
```

Checkbox.checked

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

```
document.form.checkbox.checked
```

Description

The checked property holds the current state of a check box. Because this property is a read/write Boolean, the state of the check box can be altered by changing this property.

Example

In Listing 8.43, the user is asked to select the door style wanted on her new car. When the order is submitted, the checked property of each check box is analyzed to determine whether two different door styles were selected or none at all. If two styles were selected, the check boxes are reset by altering the checked property.

Listing 8.43 Modifying the checked Property of a Check Box

```
<html>

<h2>Car Purchase Sheet</h2>

Step1: Please select the door style you want on your new car:<br>

<form name="orderForm">
    <input type="checkbox"
        name="door4">4 doors<br>
```



```

<input type="checkbox"
      name="door2">2 doors<br>
Step 2:
<input type="button"
      value="Submit Order"
      name="orderButton"
      onClick="submitOrder()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function creates a box alerting what door styles were selected.
function submitOrder()
{
    // Check for duplicate door selections
    if((document.orderForm.door4.checked == true) &&
        (document.orderForm.door2.checked == true))
    {
        // Create alert box.
        alert("You selected two different door the styles. Reselect door style.");

        // Uncheck check boxes for door styles.
        document.orderForm.door4.checked = false;
        document.orderForm.door2.checked = false;
    }
    // Check for no door selection.
    else if((document.orderForm.door4.checked == false) &&
        (document.orderForm.door2.checked == false))
    {
        // Create alert box.
        alert("You did not select a door style! Please select a door style.");
    }
    // Display the car order
    else
    {
        // Create a string to display in alert box.
        var alertString = String("Order: Car with ");
        if(document.orderForm.door4.checked == true)
            alertString += "4 doors.";          // Add 4 doors to string
        if(document.orderForm.door2.checked == true)
            alertString += "2 doors.";          // Add 2 door to string

        // Create alert box.
        alert(alertString);
    }
}
}

```

Listing 8.43 Continued

```
// -->
</script>

</html>
```

Checkbox.click()

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera5+

Syntax

```
document.form.checkbox.click()
```

Description

The `click()` method simulates the `click` event. The `clicked` property is adjusted accordingly when this method is used.

Example

In Listing 8.44, the user can select options for her new car. If the user selects the Alloy Wheel option, she is told that alloy wheels are great. If the Sports package is selected, a message is posted to let the user know that alloy wheels come with the sports package. In addition to this message, the message about alloy wheels being great is displayed and the Alloy Wheels options is checked, thanks to the `click()` method.

Listing 8.44 Using the Check Box's `click()` Method

```
<html>

<h2>Car Purchase Sheet</h2>

Step1: Please select options for your new car:<br>

<form name="orderForm">
  <input type="checkbox"
    name="alloy"
    onClick="alert('Alloy wheels look great!')">Alloy Wheels
  <br>
  <input type="checkbox"
    name="sporty"
    onClick="displayNote()">Sports Package<br>
  Step 2:
  <input type="button"
    value="Submit Order"
    name="orderButton"
    onClick="alert('Your order has been submitted')">
</form>
```

```

<script type="text/javascript" language="JavaScript">
<!--

//This function calls on another click handler
function displayNote()
{
    if(!document.orderForm.alloy.checked)
    {
        //Alert user that alloy wheels come with the sports package
        alert("Alloy wheels come with the sports package");

        //Display the alloy wheel note using click() method.
        document.orderForm.alloy.click();
    }
}
// -->
</script>

</html>

```

Checkbox.defaultChecked

JavaScript 1.0+, JScript 3.0+
 Nav2+, IE4+

Syntax

`document.form.checkbox.defaultChecked`

Description

The `defaultChecked` property holds the initial state of a check box as defined by the `checked` attribute of the `<input>` tag. Because this property only holds the initial state of the check box, it is a read-only boolean value.

Example

In Listing 8.45, the `defaultChecked` property is used to reset the car door check boxes to their initial state.

Listing 8.45 Resetting Check Boxes with the `defaultChecked` Property

```

<html>

<h2>Car Purchase Sheet</h2>

Step1: Please select the door style you want on your new car:<br>

<form name="orderForm">
  <input type="checkbox"
    name="door4">4 doors<br>

```

Listing 8.45 Continued

```
<input type="checkbox"
      name="door2"
      checked>2 doors<br>
Step 2:
<input type="button"
      value="Submit Order"
      name="orderButton"
      onClick="alert('Your order has been submitted')">
<input type="button"
      value="Reset Checkboxes"
      name="resetButton"
      onClick="resetCheckboxes()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function resets the check boxes back to their initial state
function resetCheckboxes()
{
    // Access initial state with the defaultChecked property.
    document.orderForm.door4.checked = document.orderForm.door4.defaultChecked;
    document.orderForm.door2.checked = document.orderForm.door2.defaultChecked;
}

// -->
</script>

</html>
```

Checkbox.focus()

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+

Syntax

`document.form.checkbox.focus()`

Description

The `focus()` method applies focus to a check box without invoking the check box's `onFocus` event handler.

CAUTION

In the Unix versions of Navigator 2 and Navigator 3, the `focus()` method does not work for check boxes.

Example

In Listing 8.46, two check boxes are created by using the `<input>` tag. Click the Option 1 check box and then the Option 2 check box. Finally, click the Move Focus button to move the focus back to the first check box. Do not be surprised if you do not see a difference in the check boxes in this example. In some browsers, you will not be able to detect the difference between a check box that is focused versus one that is not focused. For this reason, you will probably find very little use for this method.

Listing 8.46 Applying Focus to a Check Box Using the `focus()` Method

```
<html>
```

```
<h2>The Checkbox Focus Test</h2>
```

Click both checkboxes and then click the button to return focus to the first checkbox.

```
<form name="myForm">
  <input type="checkbox"
    name="checkBox1">Option 1<br>
  <input type="checkbox"
    name="checkBox2">Option 2<br>
  <input type="button"
    name="focusButton"
    value="Move Focus"
    onClick="moveFocus()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function moves the focus to checkBox1.
function moveFocus()
{
  // Move the focus to from checkBox1
  document.myForm.checkBox1.focus();
}

// -->
</script>

</html>
```

Checkbox.form

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

```
document.form.checkbox.form
```

Description

The `form` property provides access to a check box's parent Form object.

Example

The code in Listing 8.47 proves that the check box's `form` property contains the parent Form object by evaluating the `if` statement to true.

Listing 8.47 Accessing a Check Box's Parent with the `form` Property

```
<html>

<form name="myForm">
  <input type="checkbox"
    value="YES"
    name="myCheckBox"
    checked>
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Does the parent of the myCheckBox equal myForm?
if(document.myForm.myCheckBox.form == document.myForm)
  alert("myCheckBox's form property is equal to myForm object");
else
  alert("myCheckBox's form property is NOT equal to myForm object");

// -->
</script>

</html>
```

Checkbox.handleEvent()

JavaScript 1.2-1.3
Nav4-4.5

Syntax

```
document.form.checkbox.handleEvent(event)
```

Description

The `handleEvent()` method provides a way to invoke a check box's event handler, even though the event never happened. The *event* argument specifies the Event object associated with the event handler that is to be invoked.

Example

In Listing 8.48, the user is asked to select the engine wanted in her new car. If the user selects the V6 engine, she is alerted that the V6 has lots of power. If the 4-cylinder engine is selected, a message is posted to encourage the user to consider the V6. The click event is passed to the first check box via the `handleEvent()` method. This causes the message about more power to be displayed.

Listing 8.48 *Passing Events to a Check Box with the* `handleEvent()` *Method*

```
<html>

<h2>Car Purchase Sheet</h2>

Step1: Please select the engine style you want on your new car:<br>

<form name="orderForm">
  <input type="checkbox"
    name="v6"
    onClick="alert('The V6 has lots of power!')">V6<br>
  <input type="checkbox"
    name="cylinder4"
    onClick="displayNote(event)">4 Cylinder<br>
  Step 2:
  <input type="button"
    value="Submit Order"
    name="orderButton"
    onClick="alert('Your order has been submitted')">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function passes the click event along to another check box
// using the check box's handleEvent method.
function displayNote(event)
{
  // Display note about V6 engine
  alert("Have you considered the V6 engine?");

  // Pass event along to first V6 check box.
  document.orderForm.v6.handleEvent(event);
}

// -->
</script>

</html>
```

Checkbox.name

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

`document.form.checkbox.name`

Description

The name property provides access to the name attribute of the check box. This property is a read-only string.

Example

Listing 8.49 uses the name property of the check boxes to display what engine type was selected.

Listing 8.49 *Display the Name of the Check Box with the* `name` *Property*

```
<html>

<h2>Truck Purchase Sheet</h2>

Step1: Please select the engine style you want on your new truck:<br>

<form name="orderForm">
  <input type="checkbox"
    name="V6"
    onClick="duplicateCheck(V6,V8)">V6<br>
  <input type="checkbox"
    name="V8"
    onClick="duplicateCheck(V8,V6)">V8<br>
  Step 2:
  <input type="button"
    value="Submit Order"
    name="orderButton"
    onClick="submitOrder()">
</form>

<script type="text/javascript" language="JavaScript">
<!--

//This function uses the name property to let the user
//know what options were selected.
function submitOrder()
{
  //Create a string to be displayed in the alert box
  var alertString = String("You have selected a ");
```



```
//Determine what type of engine was selected.
if(document.orderForm.V6.checked == true)
{
    alertString += document.orderForm.V6.name; //Display V6
    alertString += " truck.";
    alert(alertString);
}
else if(document.orderForm.V8.checked == true)
{
    alertString += document.orderForm.V8.name; //Display V8
    alertString += " truck.";
    alert(alertString);
}
else
    alert("You have not selected an engine type!"); //No Engine
}

//This function removes the check from the other
//checkbox if both checkboxes are about to be selected.
function duplicateCheck(theCheckBox,otherCheckBox)
{
    //Determine if both boxes are checked
    if(theCheckBox.checked && otherCheckBox.checked)
        otherCheckBox.checked = 0;
}

// -->
</script>

</html>
```

Checkbox.onBlur

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

onBlur="command"

Description

The onBlur event handler is defined in an <input> tag and specifies what to do when a check box loses focus.

Example

The code in Listing 8.50 uses the onBlur event handler to display a message alerting the customer that the peppers she is ordering on her pizza are hot.

Listing 8.50 Use the `onBlur` Event Handler to Display an Alert Box When the Check Box Loses Focus

```
<html>

<h2>Pizza Machine</h2>

Step 1: Please select your pizza toppings:<br>

<form name="orderForm">
  <input type="checkbox"
    name="peppers"
    onBlur="pepperAlert()">Peppers<br>
  <input type="checkbox"
    name="sausage">Sausage<br>
  Step 2:
  <input type="button"
    value="Order Pizza"
    name="orderButton"
    onClick="alert('Your pizza has been ordered.')">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function alerts the customer that peppers are hot!
function pepperAlert()
{
  // If peppers are selected then display alert.
  if(document.orderForm.peppers.checked == true)
  {
    // Create alert box.
    alert("These are extremely hot peppers.");
  }
}

// -->
</script>

</html>
```

Checkbox.onClick

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera5+

Syntax

`onClick="command"`

Description

The `onClick` event handler is defined in an `<input>` tag and specifies what to do when a check box is clicked.

Example

The code in Listing 8.51 uses the `onClick` event handler to display a message alerting the customer that sausage goes well with peppers.

Listing 8.51 Use the `onClick` Event Handler to Display a Message

```
<html>

<h2>Pizza Machine</h2>

Step 1: Please select your pizza toppings:<br>

<form name="orderForm">
  <input type="checkbox"
    name="peppers"
    onClick="recommendSausage()">Peppers<br>
  <input type="checkbox"
    name="sausage">Sausage<br>
  Step 2:
  <input type="button"
    value="Order Pizza"
    name="orderButton"
    onClick="alert('Your pizza has been ordered.')">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function recommends that the customer consider ordering sausage.
function recommendSausage()
{
  // If peppers are selected, display alert.
  if(document.orderForm.peppers.checked == true)
  {
    // Create alert box
    alert("Sausage goes well with peppers.");
  }
}

// Display the event handler associated with onClick.
document.write("The pepper checkbox onClick event handler: ");
document.write(document.orderForm.peppers.onclick);
```

Listing 8.51 Continued

```
// -->
</script>

</html>
```

Checkbox.onFocus

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

```
onFocus="command"
```

Description

The onFocus event handler is defined in an <input> tag and specifies what to do when a check box gains focus.

Example

The code in Listing 8.52 uses the onFocus event handler to automatically select extra cheese when the customer selects sausage.

Listing 8.52 Using the onFocus Event Handler to Select Extra Cheese

```
<html>

<h2>Pizza Machine</h2>

Step1: Please select your pizza toppings:<br>

<form name="orderForm">
  <input type="checkbox"
    name="peppers">Peppers<br>
  <input type="checkbox"
    name="sausage"
    onFocus="chooseExtraCheese()">Sausage<br>
  <input type="checkbox"
    name="cheese">Extra Cheese<br>
  Step 2:
  <input type="button"
    value="Order Pizza"
    name="orderButton"
    onClick="alert('Your pizza has been ordered.')">
</form>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// This function chooses the extra cheese checkbox.
function chooseExtraCheese()
{
    // If sausage is selected, select extra cheese.
    if(document.orderForm.sausage.checked == false)
    {
        // select extra cheese
        document.orderForm.cheese.checked = true;
    }
}

// -->
</script>

</html>
```

Checkbox.type

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

`document.form.checkbox.type`

Description

The type property provides access to the type attribute of the check box. This property is a read-only string that always contains "checkbox".

Example

Listing 8.53 instructs the customer to select only two toppings (check boxes) of all those offered. To determine the number of toppings, the type property of the check box is used to determine how many check boxes are in orderForm. The program finds four check boxes that represent the possible pizza toppings.

Listing 8.53 Accessing a Check Box's type Property

```
<html>
<h2>Pizza Machine</h2>
```

Step 1: Please select your pizza toppings:


```
<form name="orderForm">
    <input type="checkbox"
        name="peppers">Peppers<br>
    <input type="checkbox"
        name="sausage">Sausage<br>
    <input type="checkbox"
        name="onion">Onion<br>
```

Listing 8.53 Continued

```
<input type="checkbox"
      name="bacon">Bacon<br>
Step 2:
<input type="button"
      value="Order Pizza"
      name="orderButton"
      onClick="alert('Your pizza has been ordered.')">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Initialize a counter to zero.
var counter = 0;

// Count the number of check boxes in orderForm.
for(var x=0; x<document.orderForm.length; x++)
{
    // Is element a check box?
    if(document.orderForm.elements[x].type == "checkbox")
    {
        // Increment the counter.
        counter++;
    }
}

// Display the topping instructions.
document.write("Please select no more than 2 of the ");
document.write(counter, " possible toppings.");    // Insert 4

// -->
</script>

</html>
```

Checkbox.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.form.checkbox.unwatch(property)
```

Description

The `unwatch()` method of the `Checkbox` object is used to turn off the watch for a particular property specified by *property*.

Example

Listing 8.54 shows how the `unwatch()` method is used to stop watching the *value* property of the `Checkbox` object after its value has changed to "Blue".

Listing 8.54 *Example of the* `unwatch()` *Method of the* `Checkbox` *Object*

```
<html>

<form name="myForm">
<input type="checkbox" name="myCheckbox" value="Red">
</form>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.myForm.myCheckbox.watch("value",alertme);

document.write("Original checkbox value: ");
document.write(document.myForm.myCheckbox.value + "<br>");

// Change checkbox value
document.myForm.myCheckbox.value = "Blue";

document.write("New checkbox value: ");
document.write(document.myForm.myCheckbox.value + "<br>");

// Stop watch
document.myForm.myCheckbox.unwatch("value");

// Change checkbox value
document.myForm.myCheckbox.value = "Green";

document.write("Final checkbox value: ");
document.write(document.myForm.myCheckbox.value + "<br>");

// -->
</script>
</html>
```

Checkbox.value

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

`document.form.checkbox.value`

Description

The value property provides access to value attribute of the check box. This property is a read/write value that is sent to the server when the form is submitted.

Example

Listing 8.55 uses the value property of each check box to create instructions for the customer on how to order her custom pizza.

Listing 8.55 *Accessing a Check Box's value Property*

```
<html>

<h2>Pizza Machine</h2>

Step1: Please select your pizza toppings:<br>

<form name="orderForm">
  <input type="checkbox"
    name="onion"
    value="hot onion">Onion<br>
  <input type="checkbox"
    name="bacon"
    value="spicy bacon">Bacon<br>
  Step 2:
  <input type="button"
    value="Order Pizza"
    name="orderButton"
    onClick="alert('Your pizza has been ordered.')">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Display instructions using the check box value property.
document.write("Please select either ");
document.write(document.orderForm.onion.value); // Insert hot onion
document.write(" or ",document.orderForm.bacon.value); // Insert spicy bacon
document.write(" on your custom pizza!");
```



```
// -->
</script>

</html>
```

Checkbox.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
document.form.checkbox.watch(property, function)
```

Description

The `watch()` method of the `Checkbox` object is used to turn on the watch for a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 8.56 shows how the `watch()` method is used to start watching the *value* property of the `Checkbox` object.

Listing 8.56 Example of the `watch()` Method of the `Checkbox` Object

```
<html>

<form name="myForm">
<input type="checkbox" name="myCheckbox" value="Red">
</form>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.myForm.myCheckbox.watch("value",alertme);

document.write("Original checkbox value: ");
document.write(document.myForm.myCheckbox.value + "<br>");
```

Listing 8.56 Continued

```
// Change checkbox value
document.myForm.myCheckbox.value = "Blue";

document.write("New checkbox value: ");
document.write(document.myForm.myCheckbox.value + "<br>");
// -->
</script>
</html>
```

Document

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Document object represents a Web page that is displayed in a browser window, frame, or layer. An instance is created with each document that is loaded by the browser. This object has many properties and methods that vary greatly between JavaScript and JScript. Table 8.6 lists all the properties, methods, and event handlers associated with the Document object.

Table 8.6 Properties, Methods, and Event Handlers Associated with the Document Object

Type	Item	Description
Property	alinkColor	Color of activated link.
	all	Array of all HTML tags in the document.
	anchors	Array of Anchor objects.
	applets	Array of Applet objects.
	bgcolor	Background color of document.
	classes	Style sheet classes array.
	cookie	Cookie associated with document.
	domain	Domain of document.
	embeds	Array of embedded objects.
	fgcolor	Color of text in document.
	forms	Array of Form objects.
	ids	Style sheet IDs array.
	images	Array of Image objects.
	lastModified	Date when document was last modified.
	layers	Array of Layer objects.
	linkColor	Color of links.
	links	Array of Link objects.
	plugins	Array of embedded objects.

Type	Item	Description
Method	referrer	URL of document to which the current document was linked.
	tags	Style sheet tag array.
	title	Title of document.
	URL	URL of current document.
	vlinkColor	Color of visited links.
	captureEvents()	This method captures events to be handled by a document.
	close()	This method closes output stream to a document.
	contextual()	This method applies styles to selected HTML elements.
	getSelection()	This method returns the selected text.
	open()	This method opens an output stream to a document.
	releaseEvents()	This method releases events captured by a document.
	routeEvent()	This method routes captured events to other objects.
	unwatch()	This method removes a watch point.
	watch()	This method sets a watch point.
	write()	This method appends text to a document.
	writeln()	This method appends text and a newline character to a document.
Event Handler	onClick	Handler for click events.
	onDb1Click	Handler for double-click events.
	onKeyDown	Handler for KeyDown events.
	onKeyPress	Handler for KeyPress events.
	onKeyUp	Handler for KeyUp events.
	onLoad	Handler that is used when Document has finished loading.
	onMouseDown	Handler for MouseDown events.
	onMouseUp	Handler for MouseUp events.
	onUnLoad	Handler that is used when Document unloaded from window.

document.alinkColor

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

document.alinkColor

Description

The `alinkColor` property specifies the color of activated links. A link is considered activated between the time the mouse button is clicked and released over a link. The color is expressed as a string in hexadecimal digits or as one of the HTML standard color names. The hexadecimal form is made up of six digits that follow the pattern “RRGGBB.”

Example

The sample code in Listing 8.57 sets the activated link’s color to green for all links on the page, even if they are placed before the `<script>` tags.

Listing 8.57 *Setting the Document `alinkColor` Property*

```
<html>

<a href="myGreenPage.html">The Green Site</a><br>

<script type="text/javascript" language="JavaScript">
<!--

// Set the activated links color to green.
document.alinkColor="00ff00";

// -->
</script>

<a href="myGreenGrassPage.html">The Green Grass Page</a>

</html>
```

document.all

JScript 3.0+
IE4+

Syntax

`document.all[index]`

Description

The `document.all` property is an array of all the HTML elements that are in the document. The elements appear in the array in the order in which they were created. Table 8.7 lists the methods associated with `document.all` array.

Table 8.7 *Methods Associated with the* `document.all` *Array*

Methods	Description
<code>item()</code>	Returns an HTML element based on the element's name.
<code>tags()</code>	Returns an array of elements that have the specified tag.

Example

Listing 8.58 uses the `document.all` property array and array notation (brackets) to access the *Paint* anchor, which happens to be the fourth HTML element on the page. Using dot notation, the name of the anchor is used to create a link to the top of the page.

Listing 8.58 Using `document.all`

```
<html>

<a name="Paint"><h2><u>Paint Colors</u></h2></a>
Red<br>
Green<br>
Blue<br>
Orange<br>
<br>

<script type="text/javascript" language="JavaScript">
<!--

// Create a link using the name associated with the Paint anchor (4th element).
document.write("<a href='#",document.all[4].name,"'>"); // Insert "Paint"
document.write(document.all[4].name,"</a>");           // Insert "Paint"

// -->
</script>

</html>
```

`document.all.item()`

JScript 3.0+
IE4+

Syntax

```
document.all.item(name)
```

Description

The `item()` method provides a way to retrieve an HTML element out of the `document.all` array without having to know its position in the array. Instead of using an index position, the `item()` method allows you to just pass in the name of the element as specified by the name or `id` attribute of HTML tags. Normally, the method returns

the element, but, if more than one element is found with the same name, an array of elements is returned.

Example

Listing 8.59 uses the `item()` method to access the *Paint* anchor. Using dot notation, the name of the anchor is used to create a link to the top of the page.

Listing 8.59 *Using the `item()` Method to Find a Particular HTML Element*

```
<html name="top">

<a name="Paint"><h2><u>Paint Colors</u></h2></a>
Red<br>
Green<br>
Blue<br>
Orange<br>
<br>

<script type="text/javascript" language="JavaScript">
<!--

// Create a link using the name associated with HTML element 4.
document.write("<a href='#"> // Create first part of link tag
document.write(document.all.item("Paint").name,"'>"); // Insert "Paint"
document.write(document.all.item("Paint").name,"</a>"); // Insert "Paint"

// -->
</script>

</html>
```

document.all.tags()

JScript 3.0+
IE4+

Syntax

```
document.all.tags(tag)
```

Description

The `tags()` method provides a way to retrieve all HTML elements of a particular tag type from the `document.all` array. The method returns an array of elements.

Example

Listing 8.60 uses the `tags()` method to create an array of all the anchor tags in the document. Using dot notation, the name of the first anchor in the temporary array is used to create a link to the top of the page.

Listing 8.60 Using the `tags()` Method to Find HTML Elements with a Particular Tag

```
<html>

<a name="Paint"><h2><u>Paint Colors</u></h2></a>
Red<br>
Green<br>
Blue<br>
Orange<br>
<br>

<script type="text/javascript" language="JavaScript">
<!--

// Get all the anchor tags.
var arrayOfAnchors = document.all.tags("a");

// Create a link using name of the first element in the arrayOfAnchors.
document.write("<a href='#",arrayOfAnchors[0].name,"'>");    // Insert "Paint"
document.write(arrayOfAnchors[0].name,"</a>");              // Insert "Paint"

// -->
</script>

</html>
```

document.anchors

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+, Opera5+

Syntax

```
document.anchors
document.anchors[index]
```

Description

The `anchors` property is an array that contains all the `Anchor` objects that appear within the HTML document when using the `<a name="string">` tag. The `anchors` property has one property of its own, called `length`, which contains the number of `Anchor` objects in the array. The `index` number ranges from zero to the length of the array minus one. See “Anchor,” earlier in the chapter, for a detailed explanation of all the properties associated with anchors.

CAUTION

Although hyperlinks are created using the `<a>` tag, they are not accessible through the `anchors` array. Hyperlinks are stored in the document's `links` array.

Example

Listing 8.61 demonstrates how to access anchor names using the document's anchors array.

Listing 8.61 *Accessing Anchor Names Using the* anchors *Array*

```
<html>

<a name="A"><h4>The Letter A</h4></a>
apple<br>
alligator<br>

<a name="B"><h4>The Letter B</h4></a>
baby.<br>
basketball<br>
banana<br>

<script type="text/javascript" language="JavaScript">
<!--

document.write("Anchor Names:<br>");      // Title

// Set up a loop to display the name of each anchor in document.
for(var x=0; x<=document.anchors.length; x++)
{
    // Display the name of each anchor.
    document.write(document.anchors[x].name,"<br>");
}

// -->
</script>

</html>
```

document.anchors.length

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+, Opera5+

Syntax

document.anchors.length

Description

The length property contains the number of Anchor objects that are in the document.

Example

Listing 8.62 uses the anchor length property to loop through all the anchors in the document. During each pass through the loop, a link to each anchor in the document is created.

Listing 8.62 Using the Anchor length Property to Create Hyperlinks

```

<html>

<center><h1><u>The Music Instrument Page</u></h1></center>

<br><a name="Trumpet"><h4>Trumpet</h4></a>
The trumpet is a brass instrument that can create bright, loud tones. The
trumpet has 3 valves for changing the tone being played.<br>

<br><a name="Guitar"><h4>Guitar</h4></a>
The guitar is a stringed instrument that has a hollow wooden body with a long
wooden neck. Most guitars have 6 strings each tuned to a different tone.
By pressing different combinations of strings chords can be created.<br>

<br><a name="Piano"><h4>Piano</h4></a>
The piano has one of the largest tonal ranges of any instrument. Tones
are created by pressing keys which are attached to small wood hammers that
hit strings tuned to specific tones.<br>

<script type="text/javascript" language="JavaScript">
<!--
document.write("<br>Pick an instrument:<br>");

// Create a link for each anchor using the
// Anchor object and the length property
for(var counter=0; counter<=document.anchors.length; counter++)
{
    document.write("<a href='#",document.anchors[counter].name,"'>");
    document.write(document.anchors[counter].text,"</a><br>");
}

// -->
</script>

</html>

```

document.applets

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

```
document.applets
document.applets[index]
```

Description

The applets property is an array that contains all the Applet objects that appear within the HTML document from using the <applet> tag. The applet property has one property of its own, called length, which contains the number of Applet objects in the

array. The *index* number ranges from zero to the length of the array minus one. See “Applet,” earlier in this chapter, for a detailed explanation of all the properties associated with applets.

Example

Listing 8.63 includes two fictitious calculator applets that are embedded in the HTML document. Using the `applets` array, the names of the calculators are displayed on the screen.

Listing 8.63 *Accessing Applets with the `applets` Array*

```
<html>

<applet name="Home Calculator"
        code="homeCalculator.class"
        width=50
        height=50
        mayscript></applet>
<applet name="Office Calculator"
        code="officeCalculator.class"
        width=50
        height=50
        mayscript></applet>

Special thanks goes to the individuals who
provided us with the following calculators:<br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the names of the calculator applets.
document.write(document.applets[0].name,"<br>");
document.write(document.applets[1].name);

// -->
</script>
</html>
```

document.applets.length

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

`document.applets.length`

Description

The `length` property contains the number of `Applet` objects that are in a document.

Example

Listing 8.64 uses the `length` property to display the number of applets in a HTML document.

Listing 8.64 *Accessing the Number of Applets in a Document with the `length` Property*

```
<html>

<h2>The Applets Page</h2>

<applet name="myAddApplet"
        code="add.class"
        width=50
        height=50
        mayscript></applet>
<applet name="mySubtractApplet"
        code="subtract.class"
        width=50
        height=50
        mayscript></applet>

<script type="text/javascript" language="JavaScript">
<!--

// Tell the user how many applets are currently available.
document.write("There are currently ",document.applets.length);
document.write(" applets available on this page.  Check back as");
document.write(" new applets are added daily.");
// -->
</script>

</html>
```

document.bgColor

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.bgColor`

Description

The `bgColor` property specifies the background color of the HTML document. The color is expressed as a string in hexadecimal digits or as one of the HTML standard color names. The hexadecimal form is made up of six digits that follow the pattern "RRGGBB." The color of the background can also be set with `bgcolor` attribute of the `<body>` tag.

Example

Listing 8.65 changes the document's background color based on which button is chosen.

Listing 8.65 *Modifying the Document* `bgColor` *Property*

```
<html>

<form>
<input type="button"
       value="Yellow"
       name="Yellow"
       onClick="changeBG('yellow')">
<input type="button"
       value="Green"
       name="Green"
       onClick="changeBG('green')">
<input type="text"
       name="color">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// This function changes the background color and fills in the text box.
function changeBG(color)
{
    document.bgColor=color;           // Change background color
    document.myForm.color.value=color; // Display the color
}

// -->
</script>

</html>
```

document.captureEvents()

JavaScript 1.2+
Nav4+

Syntax

```
document.captureEvents(eventMask)
```

Description

The `captureEvents()` method specifies the type of events that should be passed to the document rather than to the object for which they were intended. The *eventMask* argument(s) specifies what events to capture. The following list shows all the possible event masks. Multiple events can be captured by using the bitwise OR (`|`) operator:

- Event.ABORT
- Event.BLUR
- Event.CHANGE
- Event.CLICK
- Event.DBCLICK
- Event.DRAGDROP
- Event.ERROR
- Event.FOCUS
- Event.KEYDOWN
- Event.KEYPRESS
- Event.KEYUP
- Event.LOAD
- Event.MOUSEDOWN
- Event.MOUSEMOVE
- Event.MOUSEOUT
- Event.MOUSEOVER
- Event.MOUSEUP
- Event.MOVE
- Event.RESET
- Event.RESIZE
- Event.SELECT
- Event.SUBMIT
- Event.UNLOAD

Example

Listing 8.66 attempts to change the background color from yellow to purple when the mouse button is clicked and released. Before the Event.MOUSEDOWN and Event.MOUSEUP events can be handled by the button, they are intercepted by Document's captureEvent() method and routed to special functions that change the background colors to red and blue.

Listing 8.66 *Capture Events with the* captureEvent() *Method*

```
<html>
```

Normally the button below would toggle the background color between yellow and purple but since the mouseup and mousedown events are captured and handled by the document the events are never allowed to reach the button level.

```
<form>
<input type="button"
      value="Yellow/Purple"
      onMouseDown="document.bgColor='yellow'"
      onMouseUp="document.bgColor='purple'">
</form>
```

Listing 8.66 Continued

```
<script>
<!--

// Intercept all mouseup and mousedown events and handle them
// by document event handlers. This will cause the button event handlers
// to be intercepted.
document.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);

// Define event handlers within document to handle the mousedown
// and mouseup events.
document.onmousedown = function(event){document.bgColor='red'};
document.onmouseup = function(event){document.bgColor='blue'};

// -->
</script>

</html>
```

document.classes

JavaScript 1.2-1.3
Nav4-4.5

Syntax

`document.classes.className.tagName.style`

Description

The `classes` property is an associative array that contains classes associated with Netscape style sheets. Using dot notation, `className` specifies the `class` attribute and associated HTML tag (`tagName`) for which the `style` is applied. When `tagName` is set to `all`, the `style` is applied to all tags with a `class` attribute of `className`. The style sheet classes are created within `<style>` or `<script>` using JavaScript or HTML. The style specified can be any one of the style properties or methods shown in Table 8.8. For more information on any of the properties, see the `Style` object entries in this chapter.

NOTE

When creating a class, make sure that the declaration appears before the new class is used because many HTML objects cannot be changed after they have been created in the document.

NOTE

If JavaScript dot notation is used when creating a new class within `<style>` tags, `document` does not have to be specified.

Table 8.8 *Properties and Methods of the `Style` Object that Can Be Used with the `classes` Property*

Type	Item	Description
Property	<code>align</code>	Alignment of element within its parent
	<code>backgroundColor</code>	Background color of element
	<code>backgroundImage</code>	Background image of element
	<code>borderBottomWidth</code>	Width of bottom border of element
	<code>borderColor</code>	Color of border of element
	<code>borderLeftWidth</code>	Width of left border of element
	<code>borderRightWidth</code>	Width of right border of element
	<code>borderStyle</code>	Style of border that surrounds element
	<code>borderTopWidth</code>	Width of top border of element
	<code>clear</code>	Sides of element where floating elements are not allowed
	<code>color</code>	Foreground color of element
	<code>display</code>	Element to be displayed
	<code>fontFamily</code>	Font the element should use
	<code>fontSize</code>	Size of fonts used by element
	<code>fontStyle</code>	Font style used by element
	<code>fontWeight</code>	Font weight used by element
	<code>lineHeight</code>	Distance between two lines that are next to each other
	<code>listStyleType</code>	Format of list items elements
	<code>marginBottom</code>	Distance between bottom border of an element and top margin border of another element
	<code>marginLeft</code>	Distance between left border of an element and right margin border of another element
	<code>marginRight</code>	Distance between right border of an element and left margin border of another element
	<code>marginTop</code>	Distance between top border of an element and bottom margin border of another element
	<code>paddingBottom</code>	Distance between bottom border of element and its content
	<code>paddingLeft</code>	Distance between left border of element and its content
	<code>paddingRight</code>	Distance between right border of element and its content
	<code>paddingTop</code>	Distance between top border of element and its content
	<code>textAlign</code>	Alignment of text within element
	<code>textDecoration</code>	Type of decoration added to text
	<code>textIndent</code>	Indenting should appear before text

Table 8.8 Continued

Type	Item	Description
Method	textTransform	Transformation that should be applied to text
	verticalAlign	Vertical alignment of element
	whiteSpace	How whitespace should be handled within element
	width	Width of element
	borderWidths()	Width of border surrounding element
	margins()	Margin distance between border of an element and border of adjacent elements
	padding()	Distance between borders of element and its content

Example

Listing 8.67 demonstrates three different ways to create style sheet classes using JavaScript and HTML within <style> and <script> tags. Notice how the order in which the classes are declared and used ultimately affects the style result.

Listing 8.67 Creating New Style Sheet Classes

```
<html>

<style type="text/css">
  all.TEXTFORMAT {font-style: italic;}
</style>

<p CLASS=TEXTFORMAT>After first STYLE tag.<p>

<style type="text/javascript">
  classes.TEXTFORMAT.all.fontWeight = "bold";
</style>

<p class=TEXTFORMAT>After second STYLE tag.<p>

<script>
<!--
document.classes.TEXTFORMAT.all.textDecoration = "underline";
// -->
</script>

<p class=TEXTFORMAT>After SCRIPT tag.<p>

</html>
```


document.close()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
document.close()
```

Description

The `close()` method closes the output stream to the document. Any output that has not been written prior to calling the method will be displayed.

CAUTION

The implementation of the `close()` method varies greatly between browsers and their versions. In some browsers, output continues to be streamed after calling the `close()` method, so it is best to avoid using this method if possible. If you must use this function, test it carefully on all the browsers you intended to support.

Example

Listing 8.68 uses the `close()` method to close the output stream to a document.

Listing 8.68 Close Document Output Stream with the `close()` Method

```
<html>
```

```
<script type="text/javascript" language="JavaScript">
<!--
```

```
// Write some text to the screen and then close the document.
```

```
document.write("This line is a long line that should wrap around the ");
document.write("browser. If it does not wrap around the screen then ");
document.write("resize your browser window so that it does wrap and ");
document.write("reexecute this code.<br>");
```

```
// Close document and write all lines that are currently in the buffer.
document.close();
```

```
// Write another paragraph.
```

```
document.write("If the paragraph above wrapped around the browser then ");
document.write("you will notice that this paragraph was only written after ");
document.write("the close method was called.");
```

```
// Create an alert box so you can see where the close operation takes place.
alert("Press to continue");
```

Listing 8.68 Continued

```
// -->
</script>

</html>
```

document.contextual()

JavaScript 1.2-1.4
Nav4

Syntax

`document.contextual(context1,...,[contextN], style)`

Description

The `contextual()` method provides a way to apply a style to HTML elements based on the context of the HTML elements. For example it is possible to specify that the color of text within an `<i>` tag that appears in an `<h2>` tag is set to green. The final argument passed to the method is the *style* to be effected. The *context* arguments can also be used to specify other styles in which the final style should exist in order to be effected.

Example

The code in Listing 8.69 uses the `contextual()` method to make all italic text that appears within 2nd level header tags green.

Listing 8.69 Making All Italic Text Green with the `contextual()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
document.contextual(document.tags.h2, document.tags.i).color="green";
</script>

<h2>This word is <i>Green</i></h2>

</html>
```

document.cookie

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

`document.cookie`

Description

The `cookie` property provides the ability to read and write cookies. A cookie represents a small amount of data that a Web browser stores to allow information to be shared among Web pages.

Example

Listing 8.70 creates a cookie and then reads back the result.

Listing 8.70 Create a Cookie and Read It Back Using the `cookie` Property

```
<html>

<script>
<!--

// Create a cookie.
document.cookie = "temperature=75";

// Display the contents of the cookie.
document.write("The cookie contains: ",document.cookie);

// -->
</script>

</html>
```

document.domain

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.domain`

Description

The `domain` property initially contains the hostname of the server from which the document was loaded. The document is allowed to change the value to the domain minus the subdomain. For example, if a Web page originated from *www.example.com*, the document could change this to *example.com*. The reason this is allowed is so different pages that come from various servers within the same Web site can share properties. The restrictive quality of this property keeps unrelated documents from wrongfully seeing the data each document might have collected.

Example

No example is provided because the document would have to originate from a server that has a domain name for this property not to be empty.

document.embeds

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.embeds  
document.embeds[index]
```

Description

The `embeds` property is an array that contains all the embedded objects and plug-ins that appear within the HTML document when using the `<embed>` tag. The `embeds` property has one property of its own, called `length`, which contains the number of items in the array. The `index` number ranges from zero to the `length` minus one.

NOTE

The `embeds` array property accesses the same data as the `document.plugins` array property.

Example

Listing 8.71 uses the `length` property to display the number of embedded objects in a HTML document.

Listing 8.71 *List the Number of Embedded Objects Using the `embeds` Property*

```
<html>  
  
<h2>Learn your shapes</h2>  
  
<h2>A Circle</h2>  
<embed src="circle.gif">  
  
<h2>A Square</h2>  
<embed src="square.gif">  
  
<script type="text/javascript" language="JavaScript">  
  !..  
  
  // Display the length of the embeds array.  
  document.write(document.embeds.length, " embedded objects.");  
  
  // -->  
</script>  
  
</html>
```

document.embeds.length

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.embeds.length`

Description

The `length` property contains the number of objects that are in the `embeds[]` array.

Example

Listing 8.72 uses the `length` property to display the number of embedded objects in an HTML document.

Listing 8.72 *List the Number of Embedded Objects Using the `length` Property*

```
<html>

<h2>A Circle</h2>
<embed src="circle.gif">

<h2>A Square</h2>
<embed src="square.gif">

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the embeds array.
document.write(document.embeds.length, " embedded objects.");

// -->
</script>

</html>
```

document.fgColor

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.fgColor`

Description

The `fgColor` property specifies the default text color of all the text that appears in a Web document. This is equivalent to assigning the color to the `text` attribute in the `<body>` tag. The color is expressed as a string in hexadecimal digits or as one of the

HTML standard color names. The hexadecimal form is made up of six digits that follow the pattern “RRGGBB.”

Example

Listing 8.73 sets the default color of all the text on the page to blue.

Listing 8.73 *Set the Text Color with the `fgColor` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
  <!--

  // Set the text color to blue.
  document.fgColor="0000ff";

  // -->
</script>

<body>
  The color of all text on this page is blue.
</body>

</html>
```

document.formName

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.formName`

Description

The *formName* property is actually the name of any form that exists in the HTML document. By simply placing the name of a particular form after the word `document.`, you gain access to the specified form object. It is also possible to access forms using the `forms` property and array notation. See the `document.forms` entry in this chapter for more information.

Example

Listing 8.74 accesses the action of each form by simply specifying its name using the `document` object and dot notation.

Listing 8.74 *Accesses Form Actions Using Forms' Names*

```
<html>

<form name="Form1" action="green">
  <input type="button"
```

```

        value="Green"
        onClick = "document.bgColor='green'">
</form>
<form name="Form2" action="blue">
    <input type="button"
        value="Blue"
        onClick = "document.bgColor='blue'">
</form>

<script type="text/javascript" language="JavaScript">

// Display the action of each of the form objects.
document.write("The action associated with the Form1 object is ");
document.write(document.Form1.action, "<br>");
document.write("The action associated with the Form2 object is ");
document.write(document.Form2.action);

</script>
</html>

```

document.forms

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.forms
document.forms[index]
```

Description

The `forms` property is an array that contains all the forms that exist within the HTML document from using the `<form>` tag. The `forms` property has one property of its own, called `length`, which contains the number of items in the array. The `index` number ranges from zero to the `length` minus one.

Example

Listing 8.75 accesses the names of each form using the `forms[]` array.

Listing 8.75 Access Form Names Using the `forms` Array

```

<html>

<form name="Form1">
    <input type="button"
        value="Green"
        onClick = "document.bgColor='green'">
</form>

```

Listing 8.75 Continued

```
<form name="Form2">
  <input type="button"
    value="Blue"
    onClick = "document.bgColor='blue'">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Display the name of the form objects.
for(i=0;i<document.forms.length;i++)
{
  document.write("The name of form object ",(i+1));
  document.write(" is <i><b>",document.forms[i].name,"</b></i><br>");
}

// -->
</script>

</html>
```

document.forms.length

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

document.forms.length

Description

The length property contains the number of Form objects that are in the forms[] array.

Example

Listing 8.76 uses the length property to display the number of Form objects in the document.

Listing 8.76 Access the Number of Forms in the Document with the length Property

```
<html>

<form name="Form1">
  <input type="button"
    value="Green"
    onClick = "document.bgColor='green'">
</form>
<form name="Form2">
```



```

    <input type="button"
          value="Blue"
          onClick = "document.bgColor='blue'">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// How many items in forms[] array?
document.write(document.forms.length," Form objects in document.");

// -->
</script>

</html>

```

document.getSelection()

JavaScript 1.2+
Nav4+

Syntax

```
document.getSelection()
```

Description

The `getSelection()` method returns the text that is selected within the HTML document.

Example

Listing 8.77 uses the `getSelection()` method to display all captured text in a text box. The result of selecting "JavaScript is Great!" from the text is shown in Figure 8.3.

Listing 8.77 Displaying the Selected Text

```

<html>

The following text area will display any text that you
select within the Web page. Try selecting the phrase
"JavaScript is Great!" with the mouse.<br><br>

<form name="Form1">
  <input type="text"
        name="TextArea">
</form>

<script type="text/javascript" language="JavaScript">
<!--

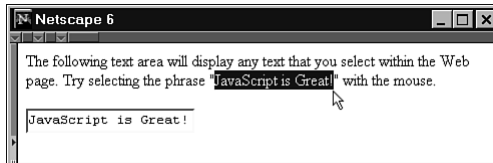
```

Listing 8.77 Continued

```
// Fill in textarea when mouse button is released.
document.captureEvents(Event.MOUSEUP);
document.onmouseup = function(event){document.Form1.TextArea.value =
➔document.getSelection()};

// -->
</script>

</html>
```

**Figure 8.3**

Selecting a portion of text and displaying it in a text box.

document.addEventListener()

JavaScript 1.2-1.3
Nav4-4.5

Syntax

`document.addEventListener(event)`

Description

The `addEventListener()` method provides a way to invoke a document's event handler, even though the event never happened. The *event* argument specifies the Event object associated with the event handler that is to be invoked.

Example

In Listing 8.78, an event handler is designated to handle all document Click events. When the user selects the button labeled 9, the `clickHandler()` method is used to route the event to the document's event handler.

Listing 8.78 Pass Events to Document with the `addEventListener()` Method

```
<html>

<h2>Math Quiz</h2>
```

What is 6+3?

```

<form name="answerForm">
  <input type="button"
    value=" 8 "
    name="answer8"
    onClick="alert('Incorrect. Try again.')">
  <input type="button"
    value=" 9 "
    name="answer9"
    onClick="document.handleEvent(event)">
</form>

<script type="text/javascript" language="JavaScript">
<!--

// Function designated to handle click events.
function clickHandler(event)
{
  // Display an alert box.
  alert("A click event occurred within the document.");
}

// Register the Click event with the document event handler.
document.onClick = clickHandler;

// -->
</script>

</html>

```

document.ids

JavaScript 1.2-1.3
Nav4-4.5

Syntax

`document.ids.idName.style`

Description

The `ids` property is an associative array that contains IDs associated with Netscape style sheets. Using dot notation, `idName` specifies an ID associated with a style. The style sheet IDs are created within the `<style>` or `<script>` tags using JavaScript or HTML. The `style` specified can be any one of the style properties or methods shown in Table 8.9. For more information on any of the properties, see the `Style` object entries in this chapter.

NOTE

When creating an ID, make sure that the declaration appears before the new ID is used because many HTML objects cannot be changed after they have been created in the document.

NOTE

If JavaScript dot notation is used when creating a new ID within `<style>` tags, document does not have to be specified.

Table 8.9 *Properties and Methods of the `Style` Object that Can Be Used with the `ids` Property*

Type	Item	Description
Property	<code>align</code>	Alignment of element within its parent
	<code>backgroundColor</code>	Background color of element
	<code>backgroundImage</code>	Background image of element
	<code>borderBottomWidth</code>	Width of bottom border of element
	<code>borderColor</code>	Color of border of element
	<code>borderLeftWidth</code>	Width of left border of element
	<code>borderRightWidth</code>	Width of right border of element
	<code>borderStyle</code>	Style of border that surrounds element
	<code>borderTopWidth</code>	Width of top border of element
	<code>clear</code>	Sides of element where floating elements are not allowed
	<code>color</code>	Foreground color of element
	<code>display</code>	Element to be displayed
	<code>fontFamily</code>	Font the element should use
	<code>fontSize</code>	Size of fonts used by element
	<code>fontStyle</code>	Font style used by element
	<code>fontWeight</code>	Font weight used by element
	<code>lineHeight</code>	Distance between two lines that are next to each other
	<code>listStyleType</code>	Format of list items elements
	<code>marginBottom</code>	Distance between bottom border of an element and top margin border of another element
	<code>marginLeft</code>	Distance between left border of an element and right margin border of another element
	<code>marginRight</code>	Distance between right border of an element and left margin border of another element
	<code>marginTop</code>	Distance between top border of an element and bottom margin border of another element

Type	Item	Description
Method	paddingBottom	Distance between bottom border of element and its content
	paddingLeft	Distance between left border of element and its content
	paddingRight	Distance between right border of element and its content
	paddingTop	Distance between top border of element and its content
	textAlign	Alignment of text within element
	textDecoration	Type of decoration added to text
	textIndent	Indenting that should appear before text
	textTransform	Transformation that should be applied to text
	verticalAlign	Vertical alignment of element
	whiteSpace	How whitespace should be handled within element
	width	Width of element
	borderWidths()	Width of border that surrounds element
	margins()	Margin distance between border of an element and margin border of adjacent elements
	padding()	Distance between borders of element and its content

Example

Listing 8.79 demonstrates three different ways to create style sheet IDs using JavaScript and HTML within `<style>` and `<script>` tags. Notice how the order in which the IDs are declared and used ultimately affects the style result.

Listing 8.79 Creating Style Sheet IDs

```

<html>

<style type="text/css">
  #TEXTFORMAT {font-style: italic;}
</style>

<p id=TEXTFORMAT>After first STYLE tag.<p>

<style type="text/javascript">
  ids.TEXTFORMAT.fontWeight = "bold";
</style>

<p id=TEXTFORMAT>After second STYLE tag.<p>

```

Listing 8.79 Continued

```
<script>
<!--
document.ids.TEXTFORMAT.textDecoration = "underline";
// -->
</script>

<p id=TEXTFORMAT>After SCRIPT tag.<p>

</html>
```

document.images

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.images
document.images[index]
```

Description

The `images` property is an array that contains all the objects that appear within the HTML document from using the `<img>` tag. The `images` property has one property of its own, called `length`, which contains the number of items in the array. The *index* number ranges from zero to the length of the array minus one.

Example

The code in Listing 8.80 accesses the source of each image using the `images` array.

Listing 8.80 Accessing Images with the `images` Array

```
<html>

<h2>A Circle</h2>


<h2>A Square</h2>
<br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the source of the image objects.
for(i=0;i<document.images.length;i++)
{
```

```

    document.write("The source of image object ",(i+1));
    document.write(" is <i><b>",document.images[i].src,"</b></i><br>");
}

// -->
</script>

</html>

```

document.images.length

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera 5+

Syntax

```
document.images.length
```

Description

The length property contains the number of objects that are in the images[] array.

Example

The code in Listing 8.81 uses the length property to display the number of images in the HTML document.

Listing 8.81 *Display the Number of Images in Document Using the length Property*

```

<html>

<h2>A Circle</h2>


<h2>A Square</h2>
<br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the images array.
document.write(document.images.length," image objects.");

// -->
</script>

</html>

```

document.lastModified

JavaScript 1.0+, JScript 3.0+
Nav2+, IE3+, Opera3+

Syntax

`document.lastModified`

Description

The `lastModified` property contains the date and time the document was last modified on the server. This property can be very useful when dealing with documents that contain information that is very date-specific. Be careful when using this date because Web servers are not required to provide this timestamp. If the timestamp is not provided, JavaScript will set the `lastModified` property to midnight, January 1, 1970 (GMT).

Example

The code in Listing 8.82 lists limited discount prices on clothing, starting with when the document was last modified.

Listing 8.82 Display Prices Based on the lastModified Property

```
<html>
<center>
<h2>LIMITED TIME SALE ON CLOTHING</h2>

<script type="text/javascript" language="JavaScript">
<!--
document.write("Starting ",document.lastModified," the following ");
document.write("clothing items will be on sale for one week ");
document.write("so order now!<br>");
// -->
</script>

<table border=0N>
  <tr>
    <th>Item</th>
    <th>Retail Price</th>
    <th>Sale Price</th>
  </tr>
  <tr>
    <td>T-shirt</td>
    <td>$20.00</td>
    <td><font color="RED">$10.99</font></td>
  </tr>
  <tr>
    <td>Jeans</td>
    <td>$60.00</td>
    <td><font color="RED">$30.99</font></td>
  </tr>
```



```

<tr>
  <td>Hats</td>
  <td>$25.00</td>
  <td><font color="RED">$15.00</font></td>
</tr>
</table>
</center>

</html>

```

document.layers

JavaScript 1.2-1.4
Nav4-4.7

Syntax

```
document.layers
document.layers[index]
```

Description

The `layers` property is an array that contains all the objects that appear within the HTML document from using the `<layer>` tag. The `layers` property has one property of its own, called `length`, which contains the number of items in the array. The *index* number ranges from zero to the length.

CAUTION

The `layers` property is no longer supported as of JavaScript 1.5, so this property is not available in Netscape 6.

Example

The code in Listing 8.83 creates two layers and then displays their names at the bottom of the page using the `layers[]` array.

Listing 8.83 Accessing the Layer Objects Using the `layers[]` Array

```

<html>

<layer name="Layer1"
  PAGEX=50
  PAGEY=50
  width=100
  height=100
  bgcolor="blue">Layer 1</layer>
<layer name="Layer2"
  PAGEX=150
  PAGEY=150

```

Listing 8.83 Continued

```
        width=100
        height=100
        bgcolor="red">Layer 2</layer>

<script type="text/javascript" language="JavaScript">
<!--

// Display the name of the layer objects.
for(i=0;i<document.layers.length;i++)
{
    document.write("The name of layer object ",(i+1));
    document.write(" is <i><b>",document.layers[i].name,"</b></i><br>");
}

// -->
</script>

</html>
```

document.layers.length

JavaScript 1.2-1.4
Nav4-4.7

Syntax

`document.layers.length`

Description

The length property contains the number of objects that are in the `layers[]` array.

CAUTION

The length property associated with the `layers[]` array is no longer supported as of JavaScript 1.5, so this property is not available in Netscape 6.

Example

Listing 8.84 creates two layers and then displays the number of layers on the bottom of the page using the length property.

Listing 8.84 Display the Number of Layers in the Document Using the length Property

```
<html>

<layer name="Layer1"
    pagex=50
    pagey=50
```

```

        width=100
        height=100
    bgcolor="blue">Layer 1</layer>
<layer name="Layer2"
    pagex=150
    pagey=150
    width=100
    height=100
    bgcolor="red">Layer 2</layer>

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the layers array.
document.write(document.layers.length," layer objects.");

// -->
</script>

</html>

```

document.linkColor

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
 Nav2+, IE3+

Syntax

```
document.linkColor
```

Description

The `linkColor` property specifies the color of unvisited links. The color is expressed as a string in hexadecimal digits or as one of the HTML standard color names. The hexadecimal form is made up of six digits that follow the pattern “RRGGBB.”

Example

Listing 8.85 sets all the unvisited links on the page to the color green.

Listing 8.85 Setting Link Colors with the linkColor Property

```

<html>

<a href="myGreenPage.html">The Green Site</a><br>

<script type="text/javascript" language="JavaScript">
<!--

```

Listing 8.85 Continued

```
// Set the unvisited links color to green.
document.linkColor="00ff00";

// -->
</script>

<a href="myGreenGrassPage.html">The Green Grass Page</a>

</html>
```

document.links

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
document.links
document.links[index]
```

Description

The `links` property is an array that contains all the `Link` objects that appear within the HTML document from using the `<a href="source">` tag. The `links` property has one property of its own, called `length`, which contains the number of `Link` objects in the array. The `index` number ranges from zero to the `length` minus one. See the `Link` object earlier in this chapter for a detailed explanation of all the properties associated with links.

CAUTION

Although anchors are created using the `<a>` tag, they are not accessible through the `links` array. Anchors are stored in the document's `anchors` array.

Example

Listing 8.86 displays the URL of each link using the `links[]` array.

Listing 8.86 Display the URL of each Link in the Document using the link array

```
<html>

<a href="EmployeeList.html">The Employee List Page</a><br>
<a href="EmployeeBenefits.html">The Employee Benefits Page</a><br>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// Display the URL of the link objects.
for(i=0;i<document.links.length;i++)
{
    document.write("The URL of link object ",(i+1));
    document.write(" is <i><b>",document.links[i].href,"</b></i><br>");
}

// -->
</script>

</html>
```

document.links.length

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

document.links.length

Description

The length property contains the number of Link objects that are in the document.links array.

Example

Listing 8.87 uses the length property to display the number of links in the HTML document.

Listing 8.87 Display the Number of Links in the Document Using the length Property

```
<html>

<a href="EmployeeList.html">The Employee List Page</a><br>
<a href="EmployeeBenefits.html">The Employee Benefits Page</a><br>

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the links array.
document.write(document.links.length," links.");

// -->
</script>

</html>
```

document.onClick

JavaScript1.0+, JScript 1.0-3.0
Nav2+, IE3-4

Syntax

`document.onClick`

Description

The `onClick` event handler specifies what should happen when the mouse is clicked within the Document object.

Example

In Listing 8.88, the script in the `<head>` of the document specifies a function to handle all `Click` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.CLICK`. When the page itself is clicked, Document's event handler generates an alert box notifying the user of the event.

Listing 8.88 Handle the Click Event with the `onClick` Event Handler

```
<html>
<head>
<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL click events
// on the page and then define a function to handle them.
document.captureEvents(Event.CLICK);
document.onClick = myClickHandler;

// Define the myClickHandler function to handle click events
function myClickHandler(e){
    alert("The document was clicked!");
}

// -->
</script>
</head>
<body>
Any time you click anywhere within this document you will
get a message alerting you that a Click event has taken place.
</body>
</html>
```

document.onDbClick

JavaScript1.0+, JScript 1.0-3.0
Nav2+, IE3-4

Syntax

document.onDbClick

Description

The onDbClick event handler specifies what should happen when the mouse is double-clicked within the Document object.

Example

In Listing 8.89, the script in the <head> of the document specifies a function to handle all DbClick events in the document. To be able to do this, the document's captureEvents() method is used to capture all events of type Event.DBCLICK. When the page itself is double-clicked, Document's event handler generates an alert box notifying the user of the event.

Listing 8.89 Handle the DbClick Event with the onDbClick Event Handler

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL DbClick events
// on the page and then define a function to handle them.
document.captureEvents(Event.DBCLICK);
document.onDbClick = myDbClickHandler;

// Define the myDbClickHandler function to handle DbClick events.
function myDbClickHandler(e){
    alert("The document was double clicked!");
}

// -->
</script>
</head>
<body>
Any time you double-click anywhere within this document, you will
get a message alerting you that a DbClick event has taken place.
</body>
</html>
```

document.onKeyDown

JavaScript1.0+, JScript 1.0-3.0

Nav2+, IE3-4

Syntax

`document.onKeyDown`

Description

The `onKeyDown` event handler specifies what should happen when any key is pressed when the Document object is in focus.

Example

In Listing 8.90, the script in the `<head>` of the document specifies a function to handle all `KeyDown` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.KEYDOWN`. When any key is pressed within the page, the document's event handler generates an alert box notifying the user of the event.

Listing 8.90 Handle the `KeyDown` Event with the `onKeyDown` Event Handler

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL key down events
// on the page and then define a function to handle them.
document.captureEvents(Event.KEYDOWN);
document.onKeyDown = myKeyDownHandler;

// Define the myKeyDownHandler function to handle
// key down events.
function myKeyDownHandler(e){
    alert("A key down event took place within the document!");
}

// -->
</script>
</head>
<body>
Anytime you press a key within this document, you will
get a message alerting you that a KeyDown event has taken place.
</body>
</html>
```


document.onKeyPress

JavaScript1.0+, JScript 1.0-3.0

Nav2+, IE3-4

Syntax

`document.onKeyPress`

Description

The `onKeyPress` event handler specifies what should happen when any key is pressed when the Document object is in focus.

Example

In Listing 8.91, the script in the `<head>` of the document specifies a function to handle all `KeyPress` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.KEYPRESS`. When any key is pressed within the page, the document's event handler generates an alert box notifying the user of the event.

Listing 8.91 Handle the `KeyPress` Event with the `onKeyPress` Event Handler

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL key press events
// on the page and then define a function to handle them.
document.captureEvents(Event.KEYPRESS);
document.onKeyPress = myKeyPressHandler;

// Define the myKeyPressHandler function to handle
// key press events.
function myKeyPressHandler(e){
    alert("A key press event took place within the document!");
}

// -->
</script>
</head>
<body>
Anytime you press a key within this document, you will
get a message alerting you that a KeyPress event has taken place.
</body>
</html>
```

document.onKeyUp

JavaScript1.0+, JScript 1.0-3.0
Nav2+, IE3-4

Syntax

`document.onKeyUp`

Description

The `onKeyUp` event handler specifies what should happen when any key is pressed and then released when the Document object is in focus.

Example

In Listing 8.92, the script in the `<head>` of the document specifies a function to handle all `KeyUp` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.KEYUP`. When any key is pressed and then released within the page, the document's event handler generates an alert box notifying the user of the event.

Listing 8.92 Handle the `KeyUp` Event with the `onKeyUp` Event Handler

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL key up events
// on the page and then define a function to handle them.
document.captureEvents(Event.KEYUP);
document.onKeyUp = myKeyUpHandler;

// Define the myKeyUpHandler function to handle
// key up events.
function myKeyUpHandler(e){
    alert("A key up event took place within the document!");
}

// -->
</script>
</head>
<body>
Anytime you press a key and release it within this document, you will
get a message alerting you that a KeyUp event has taken place.
</body>
</html>
```

document.onLoad

JavaScript1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

`onLoad="command"`

Description

The `onLoad` event handler of the Document object is fired when the page has finished loading in that particular window instance. This event handler actually belongs to the Window object but is accessible through the Document object.

NOTE

The `onLoad` event in the `<body>` of a document that is loaded in a frame will fire before an event loaded in the `<frameset>` tag that loaded the document.

Example

The sample of code in Listing 8.93 pops up an alert box when the page has finished loading using the `onLoad` event handler.

Listing 8.93 Handle the Load Event with the `onLoad` Event Handler

```
<html>
<body onLoad='alert("The document has completely loaded.")'>
<h2>The document.onLoad entry</h2>
</body>
</html>
```

document.onMouseDown

JavaScript1.0+, JScript 1.0-3.0
Nav2+, IE3-4

Syntax

`document.onMouseDown`

Description

The `onMouseDown` event handler specifies what should happen when the mouse button is clicked within the Document object.

Example

In Listing 8.94, the script in the `<head>` of the document specifies a function to handle all `MouseDown` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.MOUSEBUTTONDOWN`. When the mouse button is clicked within the page, the document's event handler generates an alert box notifying the user of the event.

Listing 8.94 *Handle the `MouseDown` Event with the `onMouseDown` Event Handler*

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL mouse
// down events on the page and then define a function
// to handle them.
document.captureEvents(Event.MOUSEDOWN);
document.onMouseDown = myMouseDownHandler;

// Define the myMouseDownHandler function to handle
// mouse down events.
function myMouseDownHandler(e){
    alert("A mouse down event took place within the document!");
}

// -->
</script>
</head>
<body>
Anytime you press the mouse button down within this document, you will
get a message alerting you that a MouseDown event has taken place.
</body>
</html>
```

document.onMouseUp

JavaScript1.0+, JScript 1.0-3.0
Nav2+, IE3-4

Syntax

`document.onMouseUp`

Description

The `onMouseUp` event handler specifies what should happen when the mouse button is clicked and then released within the `Document` object.

Example

In Listing 8.95, the script in the `<head>` of the document specifies a function to handle all `MouseUp` events in the document. To be able to do this, the document's `captureEvents()` method is used to capture all events of type `Event.MOUSEUP`. When the mouse button is clicked and then released within the page, the document's event handler generates an alert box notifying the user of the event.

Listing 8.95 Handle the MouseUp Event with the onMouseUp Event Handler

```

<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Tell the browser you want to intercept ALL mouse
// up events on the page and then define a function
// to handle them.
document.captureEvents(Event.MOUSEUP);
document.onMouseUp = myMouseUpHandler;

// Define the myMouseUpHandler function to handle
// mouse up events.
function myMouseUpHandler(e){
    alert("A mouse up event took place within the document!");
}

// -->
</script>
</head>
<body>
Anytime you press the mouse button and then release it within this
Document, you will get a message alerting you that a MouseUp
event has taken place.
</body>
</html>

```

document.onUnload

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

onUnload="command"

Description

The onUnload event handler of a Document object is fired when the page is unloaded in that particular window instance. This occurs when the user leaves the page for another page. This event handler actually belongs to the Window object but is accessible through the Document object. See Window.onUnload, later in this chapter, for more information.

NOTE

The onUnload event handler in the <body> of a document that is loaded in a frame will fire before an event loaded in the <frameset> tag that loaded the document.

Example

The code in Listing 8.96 pops up an alert box when the user clicks the bogus link to leave the current HTML page, thanks to the `onUnload` event handler.

Listing 8.96 *Handle the `onUnload` Event with the `onUnload` Event Handler*

```
<html>
<body onUnload='alert("Please do not leave!")'>
<a href="nextpage.html">Press this link to go to the next page.</a>
</body>
</html>
```

document.open()

JavaScript1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
document.open()
document.open(mimetype)
```

Description

The `open()` method of the `Document` object clears the current document and opens a stream for new data to be placed in the document. This method accepts one argument, *mimetype*, that specifies what type of data will be written to the document. The argument can be one of the following standard `mimetypes`: `text/html`, `text/plain`, `image/gif`, `image/jpeg`, or `image/x-bitmap`.

Example

Listing 8.97 opens a document with the `open()` method and then writes text to the document.

Listing 8.97 *Open a Document with the `open()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--

document.open()
document.write("Stream text to document");
document.close()

// -->
</script>
```

document.plugins

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
document.plugins
document.plugins[index]
```

Description

The `plugins` property is an array that contains all the embedded objects and plug-ins that appear within the HTML document from using the `<embed>` tag. The `plugins` property has one property of its own, called `length`, which contains the number of items in the array. The *index* number ranges from zero to the `length` minus one.

NOTE

The `plugins[]` array property accesses the same data as the `document.embeds[]` array property.

Example

Listing 8.98 uses the `length` property to display the number of embedded objects in the HTML document.

Listing 8.98 *Display the Number of Plug-ins Using the `plugins` Property*

```
<html>

<h2>Learn Shapes</h2>

<h2>A Circle</h2>
<embed src="circle.gif">

<h2>A Square</h2>
<embed src="square.gif">

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the plugins array.
document.write(document.plugins.length, " embedded objects.");

// -->
</script>

</html>
```

document.plugins.length

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`document.plugins.length`

Description

The `length` property contains the number of objects that are in the `plugins[]` array.

Example

Listing 8.99 uses the `length` property to display the number of embedded objects in the HTML document.

Listing 8.99 Display the Number of Plug-ins in the Document using the `length` property

```
<html>

<h2>A Circle</h2>
<embed src="circle.gif">

<h2>A Square</h2>
<embed src="square.gif">

<script type="text/javascript" language="JavaScript">
<!--

// Display the length of the plugins array.
document.write(document.plugins.length, " embedded objects.");

// -->
</script>

</html>>
```

document.referrer

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+

Syntax

`document.referrer`

Description

The `referrer` property contains the URL of the document that was used to reach the current document. If the URL was typed directly in to the browser's location field, this property will be empty.

Example

In Listing 8.100, the `referrer` property is used to create a link back to the previous document from within the current document in the example.

Listing 8.100 *Create a Link Back to the Calling Document Using the `referrer` Property*

```
<script type="text/javascript" language="JavaScript">
<!--

// Create a link back to the referring document.
document.write("<a href='",document.referrer,'">Go back</a>");

// -->
</script>
```

document.releaseEvents()

JavaScript1.2-1.3
Nav4-4.5

Syntax

```
document.releaseEvents(event)
document.releaseEvents(event1 | event2 | eventN)
```

Description

The `releaseEvents()` method of the Document object releases all previously captured events of the event type passed. These events can be captured with the `Document.captureEvents()` method. The following events can be released:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`
- `Event.MOUSEOVER`
- `Event.MOUSEUP`

- `Event.MOVE`
- `Event.RESET`
- `Event.RESIZE`
- `Event.SELECT`
- `Event.SUBMIT`
- `Event.UNLOAD`

After one of these events has been captured, you can define a function to replace the built-in method for handling the event. Use the `releaseEvents()` method to free the event after a capture.

Example

Listing 8.101 has a single text box and a button. The script in the `<head>` of the document specifies a function to handle all `Click` events in the document. To be able to do this, the `captureEvents()` method had to be used to capture all events of type `Event.CLICK`. When the page itself is clicked, a counter, which is displayed in the text box, is incremented.

When the mouse button is clicked, the `MouseDown` event is fired, the `Event.CLICK` is released, and the counter no longer increments when the page is clicked.

Listing 8.101 *Using the `releaseEvents()` Method to Stop Capturing Specific Events*

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
document.captureEvents(Event.CLICK);
document.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){
    // Pass all click events to the onClick event of the text box.
    document.myForm.myText.handleEvent(e);
}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}
```

```
// Releases the click event capturing
function releaseClick(){
    document.releaseEvents(Event.CLICK);
    document.onClick="";
}

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type=TEXT size=2 value="" name="myText" onClick="changeText()">
    <input type=BUTTON value="Release Event" onMouseDown="releaseClick()">
</form>
</body>
</html>
```

document.routeEvent()

JavaScript1.2+
Nav4+

Syntax

`document.routeEvent(event)`

Description

The `routeEvent()` method of the Document object passes all previously captured events of the event type passed through their normal event processes. The events that can be passed are as follows:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`
- `Event.MOUSEOVER`
- `Event.MOUSEUP`

- `Event.MOVE`
- `Event.RESET`
- `Event.RESIZE`
- `Event.SELECT`
- `Event.SUBMIT`
- `Event.UNLOAD`

After one of these events has been captured using the `Document.captureEvents()` method, you can define a function to replace the built-in method for handling the event. Use the `releaseEvents()` method to free the event after a capture, and use `routeEvent()` to allow normal processing to take place.

Example

Listing 8.102 has a single text box and a link. The script in the `<head>` of the document specifies a function to handle all `Click` events in the window. To be able to do this, the `captureEvents()` method had to be used to capture all events of type `Event.CLICK`. When the page itself is clicked, a counter, which is displayed in the text box, is incremented.

When the link is clicked, the `MouseDown` event is fired, the `Event.CLICK` is routed through its normal means, and the counter no longer increments when the page is clicked.

Listing 8.102 *Using the `routeEvent()` Method to Continue Routing a Captured Event*

```
<html>
<head>

<script type="text/javascript" language="JavaScript1.2">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
document.captureEvents(Event.CLICK);
document.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){
    // Pass all click events to the onClick event of the text box.
    document.myForm.myText.handleEvent(e);
}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
```

```
function changeText(){
    document.myForm.myText.value = counter++;
}

// Releases the click event capturing
function releaseClick(){
    document.routeEvent(Event.CLICK);
}

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="TEXT" size="2" value="" name="myText" onClick="changeText()">
    <a href="http://www.purejavascript.com"
        onMouseDown="window.routeEvent(Event.CLICK)">Click Here!</a>
</form>
</body>
</html>
```

document.tags

JavaScript 1.2+
Nav4+

Syntax

`document.tags.tagName.style`

Description

The `tags` property is an associative array that contains tags associated with Netscape style sheets. Using dot notation, `tagName` specifies a tag associated with a style. The style sheet tags are created within the `<style>` or `<script>` tags using JavaScript or HTML. The `style` specified can be any one of the style properties or methods shown in Table 8.10. For more information on any of the properties, see the `Style` object entries in this chapter.

NOTE

When creating a tag, make sure that the declaration appears before the new tag is used because many HTML objects cannot be changed after they have been created in the document.

NOTE

If JavaScript dot notation is used when creating a new tag within `<style>` tags, `document` does not have to be specified.

Table 8.10 *Properties and Methods of the `Style` Object that Can Be Used with the `tags` Property*

Type	Item	Description
Property	<code>align</code>	Alignment of element within its parent
	<code>backgroundColor</code>	Background color of element
	<code>backgroundImage</code>	Background image of element
	<code>borderBottomWidth</code>	Width of bottom border of element
	<code>borderColor</code>	Color of border of element
	<code>borderLeftWidth</code>	Width of left border of element
	<code>borderRightWidth</code>	Width of right border of element
	<code>borderStyle</code>	Style of border that surrounds element
	<code>borderTopWidth</code>	Width of top border of element
	<code>clear</code>	Sides of element where floating elements are not allowed
	<code>color</code>	Foreground color of element
	<code>display</code>	Element to be displayed
	<code>fontFamily</code>	Font the element should use
	<code>fontSize</code>	Size of fonts used by element
	<code>fontStyle</code>	Font style used by element
	<code>fontWeight</code>	Font weight used by element
	<code>lineHeight</code>	Distance between two lines that are next to each other
	<code>listStyleType</code>	Format of list items elements
	<code>marginBottom</code>	Distance between bottom border of an element and top margin border of another element
	<code>marginLeft</code>	Distance between left border of an element and right margin border of another element
	<code>marginRight</code>	Distance between right border of an element and left margin border of another element
	<code>marginTop</code>	Distance between top border of an element and bottom margin border of another element
	<code>paddingBottom</code>	Distance between bottom border of element and its content
	<code>paddingLeft</code>	Distance between left border of element and its content
	<code>paddingRight</code>	Distance between right border of element and its content
	<code>paddingTop</code>	Distance between top border of element and its content
	<code>textAlign</code>	Alignment of text within element
	<code>textDecoration</code>	Type of decoration added to text
	<code>textIndent</code>	Indenting that should appear before text

Type	Item	Description
Method	textTransform	Transformation that should be applied to text
	verticalAlign	Vertical alignment of element
	whiteSpace	How whitespace should be handled within element
	width	Width of element
	borderWidths()	Width of border that surrounding element
	margins()	Margin distance between border of an element and border of adjacent elements
	padding()	Distance between borders of element and its content

Example

Listing 8.103 demonstrates three different ways to create style sheet tags using JavaScript and HTML within `<style>` and `<script>` tags. Notice that the order in which the tags are declared and used ultimately affects the style result.

Listing 8.103 Creating New Style Sheet Tags

```

<html>

<style type="text/css">
  P {font-style: italic;}
</style>

<p>After first STYLE tag.<p>

<style type="text/javascript">
  tags.P.fontWeight = "bold";
</style>

<p>After second STYLE tag.<p>

<script>
<!--
document.tags.P.textDecoration = "underline";
// -->
</script>

<p>After SCRIPT tag.<p>

</html>

```

document.title

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`document.title`

Description

The `title` property is a read-only string that specifies the title of the document. This property is commonly set with the `<title>` tag.

Example

In Listing 8.104, the title of the Web page is written to the screen using the `title` property.

Listing 8.104 Accessing the title Property of a Document

```
<html>
<head><title>My Web Page</title></head>

<script>
<!--

// Output the title of the document.
document.write("The title of this page is <i>");
document.write(document.title,"</i>");

// -->
</script>
```

document.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`document.unwatch(property)`

Description

The `unwatch()` method of the `Document` object is used to turn off the watch for a particular property specified by *property*.

Example

Listing 8.105 shows how the `unwatch()` method is used to stop watching the *fgColor* property of the `Document` object after its value has changed to "Blue".

CAUTION

A bug associated with the `watch()` and `unwatch()` methods prevents the actual text color from changing in the browser even though the document's `fgColor` property does get changed.

Listing 8.105 Example of the `unwatch()` Method of the Document Object

```
<html>
<b>What color is this text?</b><br><br>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.watch("fgColor",alertme);

document.write("Original text color: " + document.fgColor + "<br>");

// Change text color
document.fgColor = "Blue";

document.write("New text color: " + document.fgColor + "<br>");

// Stop watch
document.unwatch("fgColor");

// Change text color
document.fgColor = "Red";

document.write("Final text color: " + document.fgColor + "<br>");

// -->
</script>
</html>
```

document.URL

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`document.URL`

Description

The `URL` property specifies the URL of the document. This property is read-only.

Example

Listing 8.106 uses the `URL` property to write the document's URL to the screen.

Listing 8.106 Accessing a Document's URL Property

```
<html>

<script>
<!--

// Output the URL of the document.
document.write("The URL of this page is -->",document.URL);

// -->
</script>

</html>
```

document.vlinkColor

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

`document.vlinkColor`

Description

The `vlinkColor` property specifies the color of visited links. The color is expressed as a string in hexadecimal digits or as one of the HTML standard color names. The hexadecimal form is made up of six digits that follow the pattern "RRGGBB."

Example

Listing 8.107 sets the visited links color to green for all links on the page, that you have visited, only if they are placed before the `<script>` tags.

Listing 8.107 *Setting the Visited Links Color with the* `vLinkColor` *Property*

```

<html>

<a href="myGreenPage.html">The Green Site</a><br>

<script type="text/javascript" language="JavaScript">
<!--

// Set the visited links color to green.
document.vlinkColor="00ff00";

// -->
</script>

<a href="myGreenGrassPage.html">The Green Grass Page</a>

</html>

```

document.watch()

JavaScript 1.2+

Nav4+, NES3+

Syntax

```
document.watch(property,function)
```

Description

The `watch()` method of the Document object is used to turn on the watch for a particular property specified by *property*. Any time the specified property is changed after the `watch()` method has been called, the specified function is called.

Example

Listing 8.108 shows how the `watch()` method is used to start watching the *fgColor* property of the Document object.

CAUTION

A bug associated with the `watch()` and `unwatch()` methods prevents the actual text color from changing in the browser even though the document's *fgColor* property does get changed.

Listing 8.108 *Example of the* `watch()` *Method of the* Document *Object*

```

<html>
<b>What color is this text?</b><br><br>

```

Listing 8.108 Continued

```
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.write(id + " changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

// Start watch
document.watch("fgColor",alertme);

document.write("Original text color: " + document.fgColor + "<br>");

// Change text color
document.fgColor = "Blue";

document.write("Final text color: " + document.fgColor + "<br>");

// -->
</script>
</html>
```

document.write()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
document.write(value,...)
```

Description

The `write()` method appends the comma-separated argument(s) (*value*) to the document as a string. If any of the arguments are not strings, they are converted to strings before being appended to the document.

Example

Listing 8.109 writes some text, as well as the value of a property, to the current document using the `write()` method.

Listing 8.109 *Displaying Text in a Document Using the write() Method*

```

<html>
<head><title>Movies</title></head>

<script type="text/javascript" language="JavaScript">
<!--

// Write data to the current document.
document.write("The title of this web page is called <u>");
document.write(document.title,"</u>");

// -->
</script>

</html>

```

document.writeln()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
document.writeln(value,...)
```

Description

The `writeln()` method appends the comma-separated argument(s) (*value*) to the document as a string. Unlike the `write()` method, the `writeln()` method appends a newline character to the document after the last argument has been written. If any of the arguments are not strings, they are converted to strings before being appended to the document.

Example

Listing 8.110 writes some text, as well as the value of a property, to the current document using the `writeln()` method. The `<pre>` tag is used to make the newline character, which was created by the `writeln()` method, appear in the Web page.

Listing 8.110 *Displaying Text on a Line in a Document Using the writeln() Method*

```

<html>
<head><title>Movies</title></head>

<script type="text/javascript" language="JavaScript">
<!--

```

Listing 8.110 Continued

```
// Write data to the current document.
document.writeln("<pre>The title of this web page is called <u>");
document.writeln(document.title,"</u></pre>");

// -->
</script>

</html>
```

Embed

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Embed object references any object that is embedded within a Web page using the HTML <embed> tag. It is inherited from the document object. Embed is typically used for audio and video files, but can be used for any type of embedded file. Embedded objects are referenced by either the embeds array or by name.

Example

Listing 8.111 shows an example of how an embedded midi file can be referenced using the embeds array.

When this HTML code is loaded in a browser, the AUTOSTART option for the <embed> tag will start playing the midi file automatically. The stopsong() function calls the stop method, which is part of browser audio plug-in, to stop playing the midi file.

Listing 8.111 Accessing an Embedded Object by the embeds Array

```
<html>

<body>
<script "type="text/javascript" language="JavaScript">
<!--
// function stops the playing of the midi song.
function stopsong(){
    document.embeds[0].stop();
}
// -->
</script>

<embed src="phantom.mid" name="phantom" width="100" height="50"
➡ autostart="true">
<br>
<form>
Click on the stop button to stop playing the midi file.
<input type="button" value="stop" onClick='stopsong()'>
</form>

</body>
</html>
```

Event

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

Core client-side JavaScript object.

Description

The Event object is a built-in object that handles the passing of properties to an event handler. The available properties are shown in Table 8.11.

Table 8.11 *Properties of the Event Object*

Type	Item	Description
Property	data	Array of URLs for dragged and dropped objects
	height	Height of the window
	layerX	Horizontal cursor position within a layer
	layerY	Vertical cursor position within a layer
	modifiers	Bit mask representing modifier keys
	pageX	Horizontal cursor position within a Web page
	pageY	Vertical cursor position within a Web page
	screenX	Horizontal cursor position within a computer screen
	screenY	Vertical cursor position within a computer screen
	target	Object for captured events
	type	Type of event
	which	The mouse button that is pressed
	width	Width of window
Method	unwatch()	Removes a watchpoint set in an event
	watch()	Adds a watchpoint to an event property

In addition to the Event properties, events exist that get handled. The available events are shown in Table 8.12.

Table 8.12 *Handled Events*

Events	Description
ABORT	Loading of Web page is interrupted by user.
BLUR	Focus is removed from the object.
CHANGE	Contents or setting for document object are changed.
CLICK	Mouse button is clicked once.
DBLCLICK	Mouse button is clicked twice.
DRAGDROP	Object is dragged and dropped.
ERROR	Error occurred during loading.
FOCUS	Focus is applied to an object.
KEYDOWN	A key is pressed down.

Events	Description
KEYPRESS	A key is pressed.
KEYUP	A key is released after being pressed down.
LOAD	Load the document within a browser.
MOUSEDOWN	The left mouse button is pressed down.
MOUSEMOVE	The mouse cursor is moved.
MOUSEOUT	The mouse cursor is moved away from a specific object.
MOUSEOVER	The mouse cursor is moved over a specific object.
MOUSEUP	The pressed mouse button is released.
MOVE	Object is moved on the screen.
RESET	Reset button is pressed.
RESIZE	Window or frame has been resized.
SELECT	Document object is selected.
SUBMIT	Submit button is pressed.
UNLOAD	Document is unloaded from the browser.

Example

Without using the Event properties, Event is used only as an argument to functions for event capturing. An example of this is shown in Listing 8.112. The example captures all KEYPRESS and DBLCLICK events for the Window object and captures SUBMIT events for the document. After the events are captured, they are passed to event handlers that perform specific functions on them. In two of the event handlers, event properties are returned.

Listing 8.112 Use of Event Keyword

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the document to capture multiple events.
window.captureEvents(Event.KEYPRESS | Event.DBLCLICK);
document.captureEvents(Event.SUBMIT);

// Function handles the KEYPRESS event.
function handlePress(evt){
    alert("You pressed a key down. The event it triggered was: " + evt.type);
    return true;
}
```

Listing 8.112 Continued

```
// Function handles the DBLCLICK event.
function handleDbClick(evt){
    alert("You double clicked at location: " + evt.pageX + "," + evt.pageY);
    return(true);
}

// Function handles the SUBMIT event.
function handleSubmit(evt){
    alert("You clicked on the submit button");
}

// This registers the :
//      handlePress function as the event handler for the KEYPRESS event
//      handleDbClick as the event handler for the DBLCLICK event
//      handleSubmit as the event handler for the SUBMIT event
window.onKeyPress = handlePress;
window.onDbClick = handleDbClick;
document.onSubmit = handleSubmit;

// -->
</script>
```

This example shows a number of different things.

```
<br><br>
```

```
<ul>
```

```
<li>How to capture multiple events</li>
```

```
<li>How to process those events using the event handlers</li>
```

```
<li>How to access properties of the <b>event</b> object</li>
```

```
</ul>
```

```
<br><br><br>
```

When you click on the submit button, it triggers the **SUBMIT** event which displays an alert box.

```
<br><br>
```

If you double click somewhere in the page, it triggers the

DBLCLICK event which displays an alert box showing the coordinates of where you double clicked.

```
<br><br>
```

When a key is pressed down in the browser, the **KEYPRESS**

event is triggered and an alert box is displayed indicating the type of event.

```
<br><br><br>
```

```
<form>
```

```
<input type="submit" value="Submit" onSubmit=''>
```

```
</form>
```

```
</body>
```

```
</html>
```

event.data

JavaScript 1.2+
Nav4+

Syntax

`event.data`

Description

The data property of the Event object references an array of strings for events of objects that have been dragged and dropped. Each string in the array contains a URL representing the dropped object. The data property can be read-only if the script has the UniversalBrowserRead privilege.

Example

Listing 8.113 shows an example of how the data property can be used to determine the URL of objects that have been dragged and dropped. The example captures all DRAGDROP events and passes them to a function called `handleDragDrop()`. The function extracts the data property from the event and outputs it.

Listing 8.113 Accessing the `event.data` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Sets up the window to capture DRAGDROP events.
window.captureEvents(Event.DRAGDROP);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handleDragDrop(evt){

    // Request the Universal Browser Read privilege.
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");

    // Declare a temporary array to hold the URL data from the event.data
    // property.
    tmp = new Array();
    tmp = evt.data;

    // Informs the user of the URL for the dragged and dropped object.
    alert("The URL for the dragdrop object is: " + tmp);
    return true;
}
```

Listing 8.113 Continued

```
// This registers the handleDragDrop function as the event handler for the
// DRAGDROP event.
window.onDragDrop = handleDragDrop;

// -->
</script>
```

This example requires the use of `<b>UniversalBrowserRead</b>` privilege.

```
<br><br>
```

Simply drag and drop an object, gif image, folder, file, etc. to the browser and an alert box will appear indicating the URL path for the object.

```
</body>
</html>
```

event.height

JavaScript 1.2+
Nav4+

Syntax

```
event.height
```

Description

The height property of the Event object controls the height of a window or frame during the RESIZE event.

Example

Listing 8.114 shows how the height property can be accessed when an event such as RESIZE occurs. The RESIZE event means that the corresponding window or frame has changed size, thereby changing the height property.

Listing 8.114 Accessing event.height Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(300,400);
}
```

```
// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("A RESIZE event has occurred. The new height of the window is: "
+ evt.height);
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;

// -->
</script>

<form name="form1">
Click button to change the window size:
<input type="button" value="Resize window" onClick = 'changeSize()'>
</form>
</body>
</html>
```

event.layerX

JavaScript 1.2+
Nav4+

Syntax

`event.layerX`

Description

The `layerX` property of the Event object controls the horizontal (x coordinate) positioning within the layer in which the event occurred.

Example

Listing 8.115 shows an example of a function that is listening for a RESIZE event. When one occurs, a variable stores the new x-coordinate position of the window in the `layerX` property. In the example, when the user resizes the window, an alert box appears informing her of the new X value.

Listing 8.115 Accessing the `layerX` Property of the event Object

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
```

Listing 8.115 Continued

```
// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(300,400);
}

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The new width (X value) after the resize is: " + evt.layerX);
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;
// -->
</script>

<form name="form1">
Click button to resize:
<input type="Button" value="Resize Window" onClick='changeSize()'>
</form>
</body>
</html>
```

event.layerY

JavaScript 1.2+
Nav4+

Syntax

`event.layerY`

Description

The `layerY` property of the `Event` object controls the vertical (y coordinate) positioning within the layer in which the event occurred. When a window or frame is resized, the new value for the vertical coordinate is stored in the `layerY` property.

Example

Listing 8.116 shows an example of how the `layerY` property is used. When a `RESIZE` event occurs, it invokes the event handler that displays an alert box indicating the new `Y` value. The `Y` value is obtained from the `layerY` property of the event object.

Listing 8.116 Example of How to Change the `layerY` Property

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(200,350);
}

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The new height (Y value) of the window object after the resize is: "
    +
    evt.layerY);
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;
// -->
</script>

<form name="form1">
Click button to resize:

<input type="Button" value="Resize Window" onClick='changeSize() '>
</form>
</body>
</html>

```

event.modifiers

JavaScript 1.2+

Nav4+

Syntax`event.modifiers`

Description

The `modifiers` property of the `Event` object refers to any keyboard modifier that occurs during an event. Modifiers are in the form of a bitmask object and might consist of the following values: `ALT_MASK`, `CONTROL_MASK`, `META_MASK`, and `SHIFT_MASK`.

Example

Listing 8.117 shows how the `modifiers` property can be accessed when some sort of modifier, such as a mouse or keyboard event, has occurred. The `KEYPRESS` event is captured and the `modifiers` property checked to see which type of key was pressed. If a match is found, a message is sent to the user so indicating.

Listing 8.117 *Accessing the `modifiers` Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Sets up the window to capture KEYPRESS events.
document.captureEvents(Event.KEYPRESS);

// function that handles the KEYPRESS event.
// It checks the event.modifiers property to see
// what button was pressed.
//
// The available values are:
//     META_MASK = 0
//     CONTROL_MASK = 2
//     ALT_MASK = 3
//     SHIFT_MASK = 4
function handlePress(evnt){

    if(evnt.modifiers == "0"){
        alert("The Meta key was pressed");
    }

    if(evnt.modifiers == "2"){
        alert("The Ctrl key was pressed");
    }

    if(evnt.modifiers == "4"){
        alert("The Shift key was pressed");
    }
    return true;
}
```



```
// This registers the handlePress function as the event handler for the
// KEYPRESS event.
document.onKeyPress = handlePress;
// -->
</script>
```

This example demonstrates the modifiers property of the event object.

The modifier checks for pressing of the Meta key.

If you press the "Ctrl" key the modifiers property indicates so.

By pressing the "Shift" key, you trigger an event which checks for the SHIFT_MASK modifier.

</body>

</html>

To access a specific modifier value, simply reference it using the Event object. Listing 8.118 shows how to access the ALT_MASK modifier.

Listing 8.118 Accessing a Specific Modifier Value

Event.ALT_MASK

event.pageX

JavaScript 1.2+

Nav4+

Syntax

`event.pageX`

Description

The `pageX` property of the Event object controls the horizontal (x coordinate) positioning within a Web page in which the event occurred.

Example

Listing 8.119 shows an example of how you can find the x-coordinate positioning of where the `click` event occurred within the browser. Simply click in the browser window and an alert box will appear indicating the x-coordinate value of where the mouse was clicked.

Listing 8.119 Example of Using the `pageX` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CLICK events.
window.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The X coordinate of where the click event occurred is: "
➤ + evt.pageX);
    return true;
}

// This registers the handle function as the event handler for the
// CLICK event.
window.onClick = handle;

// -->
</script>

<form>
This example shows you how to access the <b>pageX</b>
property of the <i>event</i> object. As you click in the
web browser, an alert box will pop up indicating the value of
the X-coordinate of where you clicked.
</form>

</body>
</html>
```

event.pageX

JavaScript 1.2+
Nav4+

Syntax

`event.pageX`

Description

The `pageY` property of the Event object controls the vertical (y coordinate) positioning within the Web page in which the event occurred.

Example

Listing 8.120 shows an example of how to determine the y-coordinate cursor positioning by using the `pageY` event property. The JavaScript code listens for a `CLICK` event to occur. When this happens, it calls the `handle()` function that is defined to handle any captured click events. The `handle()` function simply pops up an alert box indicating the y-coordinate value of where the click occurred.

Listing 8.120 *Example of Using the `pageY` Event Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CLICK events.
window.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The Y coordinate of where the click event occurred is: "
    ↪+ evt.pageY);
    return true;
}

// This registers the handle function as the event handler for the
// CLICK event.
window.onClick = handle;

// -->
</script>
```

This example shows you how to access the `pageY` property of the `event` object. As you click in the web browser, an alert box will pop up indicating the value of the Y-coordinate of where you clicked.

```
</body>
</html>
```

event.screenX

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

```
event.screenX
```

Description

The `screenX` property of the event object controls the horizontal (x coordinate) positioning within the computer screen in which the event occurred.

Example

Listing 8.121 shows an example of how to determine the x coordinate of the cursor relative to the screen of where the click event occurred. The code captures the `CLICK` event. When captured, control is passed to the `handle()` function, which determines the x-coordinate position using the `screenX` property.

Listing 8.121 Using the `screenX` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CLICK events.
window.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The X coordinate relative to the computer screen of where the
    ➤ click occurred is: " + evt.screenX);
    return true;
}

// This registers the handle function as the event handler for the
// CLICK event.
window.onClick = handle;

// -->
</script>
```

This example shows you how to access the `<b>screenX</b>` property of the `<i>event</i>` object. As you click in the web browser, an alert box will pop up indicating the value of the X-coordinate (relative to the computer screen) of where you clicked.

```
</body>
</html>
```

event.screenY

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`event.screenY`

Description

The `screenY` property of the `Event` object controls the vertical (y coordinate) positioning within the computer screen in which the event occurred.

Example

Listing 8.122 shows an example of how to determine the y coordinate of the cursor positioning relative to the computer screen when the `CLICK` event occurs. The code captures the `CLICK` event. When captured, control is passed to the `handle()` function, which determines the y-coordinate position using the `screenY` property.

Listing 8.122 Accessing the `screenY` Property of the `Event` Object

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// sets up the window to capture CLICK events
window.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The Y coordinate relative to the computer screen of where the
    ➤ click occurred is: " + evt.screenY);
    return true;
}

// This registers the handle function as the event handler for the
// CLICK event.
window.onClick = handle;

// -->
</script>
```

This example shows you how to access the `screenY` property of the `event` object. As you click in the web browser, an alert box will pop up indicating

Listing 8.122 Continued

the value of the Y-coordinate (relative to the computer screen)
of where you clicked.

```
</body>
</html>
```

event.target

JavaScript 1.2+
Nav4+

Syntax

`event.target`

Description

The `target` property of the Event object refers to the object on which the event takes place.

Example

Listing 8.123 shows an example of how the `target` property can be used to determine to which object an event occurred. In the HTML document, there are three buttons. When the button is clicked, a JavaScript function catches the `click` event and, using the `target` property, tells the user which button was clicked.

Listing 8.123 Accessing the `target` Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Informs the user which mouse button was pressed.
function whichButton(evt){
    window.captureEvents(evt.CLICK);
    alert("The button you pressed was:" + evt.target.value);
}
// -->
</script>

<form name="form1">
Choose a button and click on it.
<br><br>
<input type="button" value="Button1" name="Button1"
➤ onClick = whichButton(event)>
<input type="button" value="Button2" name="Button2"
➤ onClick = whichButton(event)>
```

```

<input type="button" value="Button3" name="Button3"
  ➤ onClick = whichButton(event)>
</form>

</body>
</html>

```

event.type

JavaScript 1.2+, JScript 3.0+
 Nav4+, IE4+

Syntax

`event.type`

Description

The `type` property of the Event object refers to the type of event that occurred. The value assigned to `type` is a string representing the name of the event. See Table 8.12 for the valid event types.

Example

Listing 8.124 shows how the `type` property can be used to figure out what type of event is being set. This example checks for a few different events. When an event is detected, a message is displayed in the message box indicating to the user which type of event occurred. This example can be expanded to include many different events.

Listing 8.124 Accessing the *type* Property

```

<html>

<body>

<form>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture multiple events.
document.captureEvents(Event.CLICK|Event.KEYPRESS|Event.MOUSEDOWN);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handle(evnt){

//alert(evnt.type)
  if(evnt.type == "click"){
    document.form1.msg.value += "The click event occurred.\n"
  }
}

```

Listing 8.124 Continued

```
    if(evnt.type == "mousedown"){
        document.form1.msg.value += "The mousedown event occurred.\n"
    }

    if(evnt.type == "keypress"){
        document.form1.msg.value += "The keypress event occurred.\n"
    }

    return true;
}

// This registers the handle function as the event handler for the
// multiple events.
document.onKeyPress = handle;
document.onClick = handle;
document.onMouseDown = handle;

// -->
</script>

<form name="form1">
This page demonstrates a few different events. Upon events occurring,
a message will be displayed in the textarea indicating which event occurred.
<br><br><br>
<ul>
<li><input type="Button" value="Click Me"></li>
<br><br>
<li>
Dummy text area.
<input type="text" size="20">
<br>
Click mouse in text field.
<br><br>
</li>

<br><br>
<b>Message output:</b>
<textarea name="msg" rows="10" cols="60"></textarea>
<br><br>
<input type="reset" value="Clear">
</form>

</body>
</html>
```


event.unwatch()

JavaScript 1.2+

Nav4+

Syntax

```
event.unwatch(prop)
```

Description

The `unwatch()` method of the Event object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, *prop*, which is the property to unwatch.

Example

Listing 8.125 shows how the `unwatch()` method is used to turn off the watchpoint set on the `height` property of the event object. When the `resize` button is clicked, the `RESIZE` event is captured. The event handler then alerts the user that the window size has changed and sets the watchpoint. A new value is then assigned to the `height` property, the watchpoint removed, and the result is displayed in the browser.

Listing 8.125 Using the `unwatch()` Method

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(300,400);
}

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + "</b>"
➤ + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("A RESIZE event has occurred. The new height of the window is: "
➤ + evt.height);
```

Listing 8.125 Continued

```
function setWatch(){
    myProp = evnt.height;
    watch("myProp",alertme);
    myProp = 200;
    unwatch("myProp");
    myProp = 100;
}

setWatch();
return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;

// -->
</script>

<form name="form1">
Click button to change the window size:
<input type="button" value="Resize window" onClick = 'changeSize() '>
</form>
</body>
</html>
```

event.watch()

JavaScript 1.2+
Nav4+

Syntax

event.watch(prop, handler)

Description

The `watch()` method of the Event object is used to watch for changes to Event properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.126 shows how the `watch()` method is used to see when the height property of the event object has changed. When the resize button is clicked, the RESIZE event is captured. The event handler then alerts the user that the window size has changed and sets the watchpoint. A new value is then assigned to the height property and then is displayed in the browser.

Listing 8.126 Using the watch() Method of the Event

```

<html>
<script type="text/javascript" language="JavaScript">
<!--
// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(300,400);
}

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" +
➤oldValue + "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("A RESIZE event has occurred. The new height of the window is: "
➤ + evt.height);

    function setWatch(){
        myProp = evt.height;
        watch("myProp",alertme);
        myProp = 200;
    }

    setWatch();
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;

// -->
</script>

<form name="form1">
Click button to change the window size:
<input type="button" value="Resize window" onClick = 'changeSize() '>
</form>
</body>
</html>

```

event.which

JavaScript 1.2+
Nav4+

Syntax

`event.which`

Description

The `which` property of the Event object refers to which key or mouse button was pressed or clicked. The value returned for mouse events is a numeric value 1, 2, or 3, representing the left, middle, and right mouse buttons, respectively. The value returned for keyboard events is a character representation for the key that was pressed.

Example

Listing 8.127 shows how the `which` property can be used to determine which mouse button was pressed. When the user clicks the radio button, an alert box is shown informing her of the corresponding number for the mouse button clicked.

Listing 8.127 Accessing the `which` Property of the Event Object

```
<html>

<body>

<form>
This example uses the which property of the event object to determine
which mouse button is pressed.
<br><br>
<input type="radio" onClick = 'alert("Mouse button Number " + event.which
➡ + "was pressed.")'>
</form>

</body>
</html>
```

event.width

JavaScript 1.2+
Nav4+

Syntax

`event.width`

Description

The `width` property of the Event object refers to the width of a window or frame. It is set during the RESIZE event to the new width of the window or frame being resized.

Example

Listing 8.128 shows an example using the width property. The RESIZE event is captured and passed to the `handle()` function. This function informs the user that a RESIZE event has occurred and outputs the new width of the window.

Listing 8.128 *Accessing the width Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that changes the size of the window.
function changeSize(){
    window.resizeTo(300,400);
}

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handle(evnt){
    alert("A RESIZE event has occurred. The new width of the window is: "
➡ + evnt.height);
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;

// -->
</script>

<form name="form1">

Click the button to resize the window:
<input type="Button" value="Resize">
</form>

</body>
</html>
```

Event.ABORT

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.ABORT

Description

The ABORT property of the Event object is used by images and refers to the event in which a transfer is interrupted or aborted by a user.

Example

Listing 8.129 shows an example in which an HTML document might have a large .gif file embedded within it, but, during the document loading process, the loading of the .gif file is aborted (by clicking the Stop button in the browser). The `captureEvents()` method catches the ABORT event and passes it to the `handleAbort()` method, which handles it accordingly.

Listing 8.129 *Using the ABORT Event Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture ABORT events.
document.captureEvents(Event.ABORT);

// function that handles the specific event. The evt parameter refers to
// the event object. In this case the function is handling any type of
// ABORT events.
function handleAbort(evt){
    alert("An ABORT event has occurred.");
    return true;
}

// This registers the handle function as the event handler for the
// ABORT event.
document.onAbort = handleAbort;
// -->
</script>
```

This page loads the `sample.gif` image. Assuming this is a large image and takes some time to load, if the user clicks the stop button on the browser, an `<b>abort</b>` event will be captured.

```
<br><br><br>
```

```


</body>
</html>
```

Event.BLUR

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.BLUR

Description

The BLUR property of the Event object is used by all windows, frames, and form elements when focus is removed from a particular object.

Example

Listing 8.130 shows an example in which the user wants to be alerted when the focus had been removed from the text area. A function is created to capture the BLUR event that occurs when focus is removed from an object. When the event is captured, the `handlerBlur()` function alerts the user of the event.

Listing 8.130 *Using the BLUR Event Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture BLUR events.
document.captureEvents(Event.BLUR);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handleBlur(evt){
    document.form1.msg.value += "A BLUR event has occurred.\n";
    return true;
}

// This registers the handle function as the event handler for the
// BLUR event.
document.onBlur = handleBlur;
```

Listing 8.130 Continued

```
// -->
</script>

<form>
Set focus to the first text box. Then click in the second
text box to remove focus from text 1.
<br><br>
Text 1:<input type="text" size="20" onBlur=''>
<br><br>
Text 2:<input type="text" size="20">
<br><br>
<b>Message box:</b>
<textarea name="msg" rows="5" cols="50"></textarea>
</form>

</body>
</html>
```

Event.CHANGE

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.CHANGE

Description

The CHANGE property of the Event object is used by any text-related and select-box form elements to indicate a change in the element settings.

Example

Listing 8.131 shows an example of a function checking for any occurrences of the CHANGE event. When a CHANGE event occurs, it is captured by the Document object and then passed to the `handleChange()` function, which alerts the user.

Listing 8.131 Accessing the CHANGE Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CHANGE event.
document.captureEvents(Event.CHANGE);
```



```
// function that handles the specific event. The evt parameter
// refers to the event object.
function handleChange(evt){
    alert("The text in TextBox1 has been changed");
    return true;
}

// This registers the handle function as the event handler for the
// CHANGE event.
document.onChange = handleChange;

// -->
</script>
```

This example demonstrates the change event. Initially TextBox1 is empty, however when you enter information into the textbox, the **CHANGE** event occurs. This triggers an alert box to open up informing you that the text in box 1 has been changed.

```
<br><br>
<form name="form1">
TextBox1:
<input type="text" size="20" name="text1" onChange=''>
<br><br>
TextBox2:
<input type="text" size="20" name="text2">
</form>
</body>
</html>
```

Event.CLICK

JavaScript 1.0+
Nav2+, Opera3+

Syntax

Event.CLICK

Description

The CLICK property of the Event object is used by all button objects, documents, and links to indicate a single mouse button click.

Example

Listing 8.132 shows how the CLICK property is used to determine whether the mouse was clicked or not. When you click the browser window, the `captureEvents()` method captures the CLICK event. The event handler then alerts you that a click has been performed.

Listing 8.132 Accessing the CLICK Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CLICK events.
window.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handleClick(evt){
    alert("A CLICK event has occurred in this window.");
    return true;
}

// This registers the handleClick function as the event handler for the
// CLICK event.
window.onClick = handleClick;

// -->
</script>

<form>
<input type="button" value="Click Here" onClick = 'handleClick(event)'">
</form>

</body>
</html>
```

Event.DBLCCLICK

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

Event.DBLCCLICK

Description

The DBLCCLICK property of the Event object is used by documents and links to indicate a double mouse click.

Example

Listing 8.133 shows an example of the onDb1C1ick event handler being used to handle the DBLCCLICK event. When the user double-clicks the button or anywhere in the window, an alert box appears indicating that she double-clicked.

Listing 8.133 Accessing the DBLCLICK event

```

<html>

<body>


<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture DBLCLICK events.
window.captureEvents(Event.DBLCLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("A DBLCLICK event has occurred in this window.");
    return true;
}

// This registers the handle function as the event handler for the
// DBLCLICK event.
window.onDbClick = handle;

// -->
</script>

```

This example demonstrates the double-click event. Double-click anywhere in the browser window and an alert box will appear indicating that a `<b>DBLCLICK</b>` event has been captured.

`<br><br>`

```

<form>
<input type="button" value="Double-click me" onDbClick=''>
</form>
</body>
</html>

```

Event.DRAGDROP

JavaScript 1.2+
Nav4+

Syntax

Event.DRAGDROP

Description

The DRAGDROP property of the Event object is used by the window to indicate the event in which an object has been dragged and dropped.

Example

Listing 8.134 shows how the DRAGDROP event is used to check for any object that has been dragged and dropped into the browser window. If this occurs, an alert box is shown indicating that the DRAGDROP event has occurred.

Listing 8.134 Using the DragDrop Event Handler

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture DRAGDROP events.
window.captureEvents(Event.DRAGDROP);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handleDragDrop(evt){
    alert("An object has been dragged and dropped.");
    return true;
}

// This registers the handle function as the event handler for the
// DRAGDROP event.
window.onDragDrop = handleDragDrop;

// -->
</script>
```

This example demonstrates the dragdrop event. Drag and drop an object in the browser window and an alert box will appear indicating that a **DRAGDROP** event has occurred.

```
<br><br>
```

```
</body>
</html>
```

Event.ERROR

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

Event.ERROR

Description

The `ERROR` property of the `Event` object is used by windows and images to indicate any errors that occurred during the loading of the Web page.

Example

Listing 8.135 shows an example of how `ERROR` events can be handled. When an `ERROR` event is captured, it is passed to the `handle()` function, which informs the user of the error.

Listing 8.135 Accessing the `ERROR` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture ERROR event.
window.captureEvents(Event.ERROR);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("ERROR: The image was unable to be loaded.");
    return true;
}

// This registers the handle function as the event handler for the
// ERROR event.
window.onerror = handle;
// -->
</script>

This page only contains a gif image. However since this gif doesn't
exist there will be an error loading the page. This error event will
be captured and alerted to the user.
<br><br>


</body>
</html>
```

Event.FOCUS

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.FOCUS

Description

The FOCUS property of the Event object is used by windows, frames, and form elements to indicate when focus is applied to an object.

Example

Listing 8.136 shows an example of how the FOCUS property is used to determine when focus is set on a certain object. As you click each object, a message is displayed in the message box indicating which object has focus. The onFocus event handler handles all Event.FOCUS events by default.

Listing 8.136 Accessing the FOCUS Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

function showMsg1(){
    document.form1.msg.value += "Focus set on Text 1.\n";
}

function showMsg2(){
    document.form1.msg.value += "Focus set on Text 2.\n";
}

function showMsg3(){
    document.form1.msg.value += "Focus set on Button 1.\n";
}

function showMsg4(){
    document.form1.msg.value += "Focus set on the Message Box.\n";
}
// -->
</script>

<form name="form1">
Set focus to an object.
<br><br>
Text 1:<input type="text" size="20" onFocus='showMsg1()'>
<br><br>
```

```

Text 2:<input type="text" size="20" onFocus='showMsg2()'>
<br><br>
Button 1:<input type="button" value="Click Me" onFocus='showMsg3()'>
<br><br>
<b>Message box:</b>
<textarea name="msg" rows="5" cols="50" onFocus='showMsg4()'></textarea>
</form>

</body>
</html>

```

Event.KEYDOWN

JavaScript 1.2+
Nav4+

Syntax

Event.KEYDOWN

Description

The KEYDOWN property of the Event object is used by documents, images, links, and text area form elements to indicate when a key is pressed by the user.

Example

Listing 8.137 shows an example of how a JavaScript function can use the KEYDOWN event to determine if a key was pressed down. When the KEYDOWN event is captured, the handle() function processes the event and informs the user that a key has been pressed down.

Listing 8.137 Accessing the KEYDOWN Property

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture KEYDOWN events.
window.captureEvents(Event.KEYDOWN);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    document.form1.msg.value += " A key was pressed down.\n";
    return true;
}

```

Listing 8.137 Continued

```
// This registers the handle function as the event handler for the
// KEYPRESS event.
window.onKeyDown = handle;
// -->
</script>
<form name="form1">
```

Press a key down. When a key is pressed down, a message is displayed in the message box indicating that the `<b>KEYDOWN</b>` event has occurred.

```
<br><br>
<b>Message box:</b>
<textarea name="msg" rows="5" cols="50"></textarea>
<br><br>
<input type="reset" value="Clear Message Box">
</form>

</body>
</html>
```

Event.KEYPRESS

JavaScript 1.2+
Nav4+

Syntax

Event.KEYPRESS

Description

The `KEYPRESS` property of the `Event` object is used by documents, images, links, and text area form elements to indicate when a key is pressed and held by the user.

Example

Listing 8.138 shows how the `KEYPRESS` property is used to determine when a key has been pressed. When the `KEYPRESS` event is captured, it is sent to the `handle()` function, which informs the user that a key has been pressed.

Listing 8.138 Accessing the `KEYPRESS` Object

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture KEYPRESS events.
window.captureEvents(Event.KEYPRESS);
```



```
// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    document.form1.msg.value += " A key was pressed.\n";
    return true;
}

// This registers the handle function as the event handler for the
// KEYPRESS event.
window.onKeyPress = handle;
// -->
</script>

<form name="form1">
Press a key. When a key is pressed, a message is displayed in the message box
indicating that a key has been pressed.
<br><br>
<b>Message box:</b>
<textarea name="msg" rows="5" cols="50"></textarea>
</form>

</body>
</html>
```

Event.KEYUP

JavaScript 1.2+
Nav4+

Syntax

Event.KEYUP

Description

The KEYUP property of the Event object is used by documents, images, links, and text area form elements to indicate when a pressed key is released by the user.

Example

Listing 8.139 shows how to determine when a key has been released. The KEYUP event is captured. When this occurs, the handle() function is invoked, informing the user that a key which had been pressed down has been released.

Listing 8.139 Accessing the KEYUP Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
```

Listing 8.139 Continued

```
<!--

// Sets up the window to capture KEYUP events.
window.captureEvents(Event.KEYUP);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    document.form1.msg.value += " A key was pressed down and let up.\n";
    return true;
}

// This registers the handle function as the event handler for the
// KEYPRESS event.
window.onKeyUp = handle;
// -->
</script>

<form name="form1">

Press a key down. When a key is pressed down and let back up, a message is
displayed in the message box indicating that the <b>KEYUP</b> event has
occurred.If you press a key down and hold it down, no message displayed.
Only when the key
is let back up is a message displayed.
<br><br>
<b>Message box:</b>
<textarea name="msg" rows="5" cols="50"></textarea>
<br><br>
<input type="reset" value="Clear Message Box">
</form>

</body>
</html>
```

Event.LOAD

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

Event.LOAD

Description

The LOAD property of the Event object is used by the Document object to indicate when a page is loaded by the browser.

Example

Listing 8.140 shows an example of how to inform the user when a Web page has loaded. The LOAD event is captured. When this occurs, the `handle()` function is invoked, which informs the user that the page has finished being loaded.

Listing 8.140 *Example of Using the LOAD Event Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
// Sets up the window to capture LOAD events.
window.captureEvents(Event.LOAD);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handle(evnt){
    alert("The page is finished being loaded.");
    return true;
}

// This registers the handle function as the event handler for the
// LOAD event.
window.onload = handle;

// -->
</script>
```

This page only contains a gif image. When the page is finished loading, a message is displayed indicating so.

```
<br><br>

</body>
</html>
```

Event.MOUSEBUTTONDOWN

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

Event.MOUSEBUTTONDOWN

Description

The MOUSEBUTTONDOWN property of the Event object is used by button objects, documents, and links to indicate when the mouse button is pressed by the user.

Example

Listing 8.141 shows how to use the `MOUSEDOWN` property to determine when the mouse has been pressed down. The `onMouseDown` event handler is used to catch the `MOUSEDOWN` event. When it is caught, an alert box is used to inform the user that the event has just occurred.

Listing 8.141 *Accessing the* `MOUSEDOWN` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture MOUSEDOWN events.
window.captureEvents(Event.MOUSEDOWN);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The mouse button has been pressed down.");
    return true;
}

// This registers the handle function as the event handler for the
// MOUSEDOWN event.
window.onMouseDown = handle;

// -->
</script>
```

This example uses the `<b>MOUSEDOWN</b>` event. When the mouse button is pressed down, an alert box is shown.

```
</body>
</html>
```

Event.MOUSEMOVE

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`Event.MOUSEMOVE`

Description

The `MOUSEMOVE` property of the `Event` object indicates when the mouse cursor is moved by the user.

Example

Listing 8.142 shows how the MOUSEMOVE event is used to determine when the user is moving the mouse. As the mouse cursor is moved, the coordinates are displayed in the text boxes.

Listing 8.142 *Accessing the* MOUSEMOVE *Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture MOUSEMOVE events.
window.captureEvents(Event.MOUSEMOVE);

// function that handles the specific event. The evnt parameter refers to
// the event object. The function sets the x and y coordinates into the text
// areas.
function handle(evnt){
    document.form1.x.value=evnt.pageX;
    document.form1.y.value=evnt.pageY;
    return true;
}

// This registers the handle function as the event handler for the
// MOUSEMOVE event.
window.onMouseMove = handle;
// -->
</script>

<form name="form1">
This example uses the <b>MOUSEMOVE</b> event. When the mouse is moved,
then the coordinates are changed.
<br><br>
Mouse x-Coordinate value:<input type="text" name="x" size="3"><br><br>
Mouse y-Coordinate value:<input type="text" name="y" size="3">
</form>
</body>
</html>
```

Event.MOUSEOUT

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

Event.MOUSEOUT

Description

The `MOUSEOUT` property of the `Event` object is used by links and document layers to indicate when the focus of the mouse cursor is moved away from an object.

Example

Listing 8.143 shows how the `onMouseOut` event handler is used to catch the `MOUSEOUT` event that occurs when the mouse cursor is removed from a HTML link.

Listing 8.143 *Using the `MOUSEOUT` Property*

```
<html>

<body>

<a href="http://www.microsoft.com" onMouseout = 'alert("The mouse has moved
➡ out of the area of this link")'>
Microsoft Website</a>

</body>
</html>
```

Event.MOUSEOVER

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

property of the `Event` object is used by links and document layers to indicate when the mouse cursor is moved over an object.

Example

Listing 8.144 shows an example of how the `MOUSEOVER` event property can be used to modify form element values. As the user moves the mouse cursor over the link, the `MOUSEOVER` event is captured and the user is alerted.

Listing 8.144 *Accessing the `MOUSEOVER` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture MOUSEOVER events.
window.captureEvents(Event.MOUSEOVER);
```

```
// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("Your mouse cursor is over the Netscape link.");
    return true;
}

// This registers the handle function as the event handler for the
// MOUSEOVER event.
window.onMouseOver = handle;

// -->
</script>

<form name="form1">
Move the mouse cursor over the link to Netscape. When you do,
it will trigger a <b>MOUSEOVER</b> event which is captured.
An alert box will then appear indicating that the mouse cursor
is over the link.
<br><br><br>
<a href="http://www.netscape.com" onMouseOver = ''>
Link to Netscape Website</a>
<br>
<br><br><br></form>

</body>
</html>
```

Event.MOUSEUP

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

Event.MOUSEUP

Description

The MOUSEUP property of the Event object is used by button objects, documents, and links to indicate when a mouse button is released.

Example

Listing 8.145 shows how the onMouseUp event handler is used to determine when the mouse button is released after being pressed. The user clicks the button and, when the mouse button is released, a message is displayed indicating the action.

Listing 8.145 Accessing the `MOUSEUP` Property

```
<html>

<body>

<form>
This example demonstrates the MOUSEUP event. When you click the
button and let the mouse up, an alert message is
displayed indicating that the event occurred.
<input type="button" value="Click Me"
onMouseup = 'alert("The Mouse button was let up")'>
</form>

</body>
</html>
```

Event.MOVE

JavaScript 1.2+
Nav4+

Syntax

Event.MOVE

Description

The `MOVE` property of the `Event` object is used by windows and frames to indicate when movement by the window or frame occurs.

Example

Listing 8.146 shows the syntax for accessing the `MOVE` property. When you begin to move the browser window, the `MOVE` event will be captured, and the coordinates for the upper-left of the window will be displayed.

Listing 8.146 Accessing the `MOVE` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture MOVE events.
window.captureEvents(Event.MOVE);

// function that handles the specific event. The evnt parameter refers to
// the event object. The function sets the x and y coordinates into the text
// areas.
```



```

function handleMove(evt){
    document.form1.msg.value="The window has been moved to coordinates: "
    + evt.screenX + "," + screenY;
    document.form1.x.value=evt.screenX;
    document.form1.y.value=evt.screenY;
    return true;
}

// This registers the handleMove function as the event handler for the
// MOVE event.
window.onMove = handleMove;

// -->
</script>

<form name="form1">
This example uses the <b>MOVE</b> event. When the browser window is moved,
then the coordinates are displayed and updated in the field.
<br><br>
Upper-Left Corner X-Coordinate value:
<input type="text" name="x" size="3"><br><br>
Upper-Left Corner Y-Coordinate value:
<input type="text" name="y" size="3">
<br><br><br>
Message Box:
<br>
<textarea name="msg" rows="3" cols="60"></textarea>
<br>
<input type="reset" value="Clear">
</form>

</body>
</html>

```

Event.RESET

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

Event.RESET

Description

The RESET property of the Event object is used solely by forms to indicate when the Reset button is clicked.

Example

Listing 8.147 shows an example of how to determine if a Reset button has been pressed. When the RESET event is captured, the function `handle()` alerts the user that the button has been clicked.

Listing 8.147 *Accessing the RESET Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture RESET events.
window.captureEvents(Event.RESET);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("You have clicked the Reset button");
    return true;
}

// This registers the handle function as the event handler for the
// RESET event.
window.onReset = handle;
// -->
</script>
```

This example demonstrates the `<b>Reset</b>` event.

- Click the different buttons below. Only the Reset
- button will trigger the submit event.

```
<br><br>
<form>
<br><br>
Dummy Text:<input type="text" size="20">
<br><br>
<input type="button" value="Click Button 1">
<br><br>
<input type="button" value="Click Button 2">
<br><br>
<input type="reset" value="Reset" onReset=''>
</form>
</body>
</html>
```

Event.RESIZE

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

Event.RESIZE

Description

The `RESIZE` property of the Event object is used by windows and frames to indicate the event of resizing the window or frame.

Example

Listing 8.148 shows how to determine when the window has been resized using the `RESIZE` event. When the `RESIZE` event is captured, the `handle()` function outputs the height and width properties of the window.

Listing 8.148 *Using the RESIZE Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture RESIZE events.
window.captureEvents(Event.RESIZE);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("You have resized the window to: " + evt.height + "x" + evt.width);
    return true;
}

// This registers the handle function as the event handler for the
// RESIZE event.
window.onResize = handle;
// -->
</script>

<form name="form1">
Click button to resize:
<input type="Button" value="Resize" onClick=window.resizeTo(250,400)>
</form>

</body>
</html>
```

Event.SELECT

JavaScript 1.0+
Nav2+, Opera3+

Syntax

Event.SELECT

Description

The `SELECT` property of the `Event` object is used by text objects and select-box form elements to indicate when an element is selected by the user.

Example

Listing 8.149 shows how the `SELECT` event can be used to determine which form object has been selected. When some text in the `textarea` is selected, the `SELECT` event is invoked. The document will then capture the event and call the `handleSelect()` function, which informs the user that the `SELECT` event occurred.

Listing 8.149 *Using the SELECT Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture SELECT events.
document.captureEvents(Event.SELECT);

// Handles the SELECT event.
function handleSelect(evt){

    alert("You have selected some text in the textarea");
}

// This registers the handle function as the event handler for the
// SELECT event.
document.onSelect = handleSelect;

// -->
</script>

<form name="form1">
Message:
<input type="text" size="25" name="txtbox" value="" onSelect=''>
<br><br><br>
Display text:<br>
<textarea name="msg" rows="5" cols="60" onSelect=''>
```

In this example we check for the SELECT event to occur. This occurs when some text in a textarea or textbox is selected. To see this, select some text in this paragraph and an alert box will pop up indicating that the select event has occurred.

```
</textarea>
<br><br>
</form>
```

```
</body>
</html>
```

Event.SUBMIT

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.SUBMIT

Description

The SUBMIT property of the Event object is used solely by forms to indicate the clicking of the Submit button.

Example

Listing 8.150 shows an example of how JavaScript can be used to process a form when the Submit button is clicked. The Window listens for the SUBMIT event. When captured, it will call the handle() function. This function informs the user that the Submit button was clicked.

Listing 8.150 *Example of Using the SUBMIT Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture SUBMIT events.
window.captureEvents(Event.SUBMIT);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("You have clicked the Submit button");
    return true;
}
```

Listing 8.150 Continued

```
// This registers the handle function as the event handler for the
// SUBMIT event.
window.onSubmit = handle;
// -->
</script>
```

This example demonstrates the submit event. Click the different buttons below. Only the Submit button will trigger the submit event.

```
<br><br>
<form>
<input type="button" value="Click Me">
<br><br>
<input type="Submit" value="Submit" onSubmit=''>
<br><br>
<input type="button" value="Click on Me too">
</form>
</body>
</html>
```

Event.UNLOAD

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Event.UNLOAD

Description

The UNLOAD property of the Event object is used by documents to indicate when a new document is loaded in a browser or when the browser window is closed.

Example

Listing 8.151 shows how the UNLOAD property can be used to find out when a browser has finished unloading a Web page. When the UNLOAD event has been captured, the `handle()` function is called, which informs the user that the page has been unloaded.

Listing 8.151 Accessing the UNLOAD Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture UNLOAD events.
window.captureEvents(Event.UNLOAD);
```

```
// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The page has been unloaded.");
    return true;
}

// This registers the handle function as the event handler for the
// LOAD event.
window.onunload = handle;

// -->
</script>
```

This page only contains a gif image. When it is unloaded, an alert message is displayed indicating so.<i> (To unload the page, simply load another page or click on the browser back button.)</i>

```
<br><br>


</body>
</html>
```

FileUpload

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The FileUpload object represents a file upload box within an HTML form. An upload box is created by using the HTML <input> tag and specifying the type attribute as file. The FileUpload object has specific properties and methods associated with it, which are shown in Table 8.13.

Table 8.13 *Properties and Methods of the FileUpload object*

Type	Item	Description
Property	form	Reference form object containing the FileUpload box
	name	HTML name attribute for the FileUpload box
	onBlur	Event handler for the Blur event
	onChange	Event handler for the Change event
	onFocus	Event handler for the Focus event

Table 8.13 Continued

Type	Item	Description
Method	type	The HTML type attribute for the FileUpload box
	value	String specifying the pathname of a selected file
	blur()	Removes focus from the FileUpload box
	focus()	Sets focus on the FileUpload box
	handleEvent()	Handles specific event
	select()	Selects input area for the FileUpload box
	unwatch()	Removes a watchpoint from a FileUpload property
	watch()	Sets a watchpoint on a FileUpload property

Example

Listing 8.152 shows how an upload box is created and then how the name property is accessed using the FileUpload object.

A FileUpload object in the HTML page contains a Browse button that allows you to browse the computer for a file to upload. After this is chosen, normally it would be sent to a server to be uploaded. However, this example only demonstrates how to get the full pathname for the file to be uploaded.

Listing 8.152 How the FileUpload Object Is Used

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function demonstrates how to obtain property values of the
// FileUpload object.
function showname(){
    // Declare a variable to hold the name of the upload box.
    var file = document.form1.uploadBox.value ;
    document.form1.filename.value = file ;
}
// -->
</script>

<form name="form1">
Click on browse to choose a file to send.
<br>
Click on the Send button to see the full path for the file sent.
<br><br>
File to send: <input type="file" name="uploadBox">
<br><br>
<input type="button" value="Send" name="get" onClick='showname()'>
<br><br>
<input type="text" name="filename" size="40">
```



```

</form>
</body>
</html>

```

FileUpload.blur()

JavaScript 1.1+
Nav3+, Opera3+

Syntax

```
fileupload.blur()
```

Description

The `blur()` method of the `FileUpload` object is used to remove focus from the `FileUpload` box.

Example

Listing 8.153 shows how the `blur()` method is used to remove focus from the upload box. When the OK button is clicked, the focus is removed from the upload box and a message is displayed.

Listing 8.153 Example of the `blur()` Method

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function showMessage(){
// Removes focus from the upload box and writes text.
    document.form1.uploadbox.blur();
    document.form1.textbox.value = "File Submitted";
}
// -->
</script>

<form name="form1">
Enter Filename:
<input type="file" name="uploadbox">
<input type="button" value="Okay" onClick=showMessage()>
<br><br>
Confirmation:
<input type="text" name="textbox">

</form>

</body>
</html>

```

FileUpload.focus()

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

```
fileupload.focus()
```

Description

The `focus()` method is used to set focus to the `FileUpload` object.

Example

Listing 8.154 shows how to set the focus on the `FileUpload` object. When the user clicks the OK button, the JavaScript function `checkFile()` is called to reset the focus to the upload box and display a message.

Listing 8.154 Setting Focus to the Upload Box

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function checkFile(){
// Sets focus to the upload box.
    document.form1.uploadbox.focus();
    document.form1.textbox.value = "Verify that filename is correct";
}
// -->
</script>

<form name="form1">
Enter Filename:
<input type="file" name="uploadbox">
<input type="button" value="Okay" onClick=checkFile()>
<br><br>
Confirmation Message:
<input type="text" name="textbox" size="35">

</form>

</body>
</html>
```

FileUpload.form

JavaScript 1.0+, JScript 3.0+
Nav2+, IE4+, Opera3+

Syntax

`fileupload.form`

Description

The `form` property() of the `FileUpload` object is used to reference the form object that contains the `FileUpload` box.

Example

Listing 8.155 shows an example of how the `form` property can be used to extract any attributes of the form containing the `FileUpload` box. The page contains two boxes in which a file can be specified to be uploaded. The `checkFiles()` function verifies whether a file has been chosen for each upload box. If a file hasn't been chosen, the script will alert the user.

Listing 8.155 Using the `FileUpload` form Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function verifies whether a file has been chosen for each FileUpload box.
function checkFiles(){

    if (document.secret.file1.value == ""){
        alert("You did not enter anything for file 1");
    }
    if (document.secret.file2.value == ""){
        alert("You did not enter anything for file 2");
    }
    else {
        alert("The files are okay and will be uploaded");
    }
}
// -->
</script>

<form name="secret">

Please choose two files to upload.
<br><br>
File 1:<input type="file" name="file1">
<br><br>
```

Listing 8.155 Continued

```
File 2:<input type="file" name="file2">
<br><br>
<input type="button" value="Verify" onClick='checkFiles()''>
</form>

</body>
</html>
```

The first example shows one method of referencing the upload box. There is a second way, though. It can also be referenced by using the form elements array. An example of this is in Listing 8.156.

Listing 8.156 Second Method of Referencing a FileUpload Object Using the forms Elements Array

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function showUploadName(){
    alert("The FileUpload box name is: " + document.secret.elements[0].name);
}
// -->
</script>

<form name="secret">
Please choose a file to be uploaded.
<br><br>
<input type="file" name="mybox" >
<br><br>
Click the button to get the name of the form containing the FileUpload box.
<br><br>
<input type="button" value="Get Form Name" onClick='showUploadName()''>
</form>

</body>
</html>
```

FileUpload.handleEvent()

JavaScript 1.2+
Nav4+

Syntax

```
fileupload.handleEvent(event)
```

Description

The `handleEvent()` method of the `FileUpload` object invokes the event handler for the specific event.

Example

Listing 8.157 shows how the `handleEvent()` method is used to handle all `CHANGE` events. When the user chooses or enters a filename in the `FileUpload` box and then changes the information, the `CHANGE` event occurs and is captured. The `handleChange` function processes the `CHANGE` event and passes it to the `handleEvent()` method of the upload box. So the event handler for the upload box will handle all `CHANGE` events.

Listing 8.157 *Using the* `handleEvent()` *Method*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Sets up the window to capture CHANGE events.
window.captureEvents(Event.CHANGE);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handleChange(evnt){
    window.document.uploadbox.handleEvent(evnt);
}

function displayText(){
    document.form1.msg.value += "Change made to object\n";
}

// This registers the handle function as the event handler for the
// CHANGE event.
window.onChange = handleChange;
// -->
</script>

Choose a file:
<br>
<form name="form1">
<input type="file" size="40" name="uploadbox" onChange='displayText()'>
<br><br>
<textarea name="msg" rows="10" cols="50"></textarea>
```

Listing 8.157 Continued

```
</form>

</body>
</html>
```

FileUpload.name

JavaScript 1.0+
Nav2+, Opera3+

Syntax

fileupload.name

Description

The name property of the FileUpload object represents the name attribute of the HTML `<input>` tag that creates the FileUpload box. This allows you to reference a FileUpload object directly by name.

Example

Listing 8.158 shows how the name of the upload box is used to access its properties. The function `getname()` uses the form object and the name of the upload box to access the name property.

Listing 8.158 Accessing the FileUpload Object by Name

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Function alerts the user to what the name of the upload box is.
function getname(){
    var boxname = document.form1.myUploadbox.name;
    alert("The name of the FileUpload box is: " + boxname);
}
// -->
</script>

<form name="form1">
Click on the button below to get the name of the upload box.
<br><br>
<input type="file" name="myUploadbox">
<br><br>
```

```

<input type="button" value="Get Name" onClick='getname() '>
</form>

</form>
</html>

```

FileUpload.onBlur

JavaScript 1.1+
Nav3+, Opera3+

Syntax

onBlur="command"

Description

The onBlur event handler is an event handler for the FileUpload object that notifies you when the focus is removed from an upload box.

Example

Listing 8.159 shows how the onBlur event handler is used to detect when the focus is removed from the specified upload box. The user chooses a file from the FileUpload box and then clicks the text box that removes the focus from the FileUpload box, causing the Blur event to be thrown.

Listing 8.159 Example of the FileUpload onBlur Method

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function inform(){
    document.form1.msg.value="File submitted and focus removed from
    FileUpload object";
}

// -->
</script>

<form name="form1">
Please choose a file to upload to the server.
<br><br>
<input type="file" onBlur= 'inform()'>
<br><br>
Click on the text box.
<br><br>

```

Listing 8.159 Continued

Message:

```
<input type="text" name="msg" size="50">
</form>

</body>
</html>
```

FileUpload.onChange

JavaScript 1.1+

Nav3+, Opera3+

Syntax

`onChange="command"`

Description

The `onChange` event handler of the `FileUpload` object is an event handler that notifies you when the upload box information has been changed.

Example

Listing 8.160 uses the `onChange` event handler to check for a user entering information into an upload box. When the filename entered has been changed, the `onChange` event handler is triggered and a message is displayed in the text box.

Listing 8.160 Example of the `onChange` Event Handler

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// function that informs the user that the filename in the FileUpload box
// has been changed.
function inform(){
    document.form1.msg.value = "Filename has been changed";
}

// -->
</script>

<form name="form1">

Please choose a file.
<input type="file" name="uploadbox" size="35" onChange='inform()'>
<br><br>
```


Message:

```
<input type="text" name="msg" size="40">
</form>
```

```
</body>
</html>
```

FileUpload.onFocus

JavaScript 1.1+
Nav3+, Opera3+

Syntax

```
onFocus="command"
```

Description

The onFocus event handler of the FileUpload object notifies you when the focus is set on the upload box.

Example

In Listing 8.161, the onFocus event handler is used to notify the user when the focus is moved to the upload box. If the user sets the focus to the FileUpload box, a message is displayed. If the user removes the focus from the FileUpload box, another message is displayed.

Listing 8.161 Example of the onFocus Event Handler

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// function that displays a message whenever focus is set on the FileUpload
// box.
function showMsg1(){
    document.form1.msg.value = "Focus on the FileUpload box\n";
}

// Function displays a message when focus is removed from the FileUpload box.
function showMsg2(){
    document.form1.msg.value = "Focus removed from FileUpload box\n";
}

// -->
</script>
<form name="form1">
```

Listing 8.161 Continued

Click in the FileUpload box to set focus to it.

```
<br><br>
<input type="file" name="uploadbox" onFocus = 'showMsg1()'>
<br><br>
Click on the button to remove focus from the FileUpload box.
<br>
<input type="button" value="Click here" onClick='showMsg2()'>
<br><br>
<textarea name="msg" rows="5" cols="50"></textarea>
</form>

</body>
</html>
```

FileUpload.select()

JavaScript 1.0+
Nav2+, Opera3+

Syntax

```
fileupload.select()
```

Description

The select() method of the FileUpload box is used to select the input area of the upload field.

Example

Listing 8.162 shows an example of how the select() method is used to select the input text box of the FileUpload box.

Listing 8.162 Using the select() Method of the FileUpload Object

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function enterName(){
// Selects the input area of upload box when button is clicked.
    document.form1.uploadbox.select();
}
// -->
</script>

<form name="form1">
<input type="file" name="uploadbox">
<br><br>
```

```
<input type="button" value="Go to Filename Box" onClick=enterName()>

</body>
</html>
```

FileUpload.type

JavaScript 1.1+
Nav3+, Opera3+

Syntax

fileupload.type

Description

The type property of the FileUpload object represents the type attribute of the HTML <input> tag used to create the upload box.

Example

Listing 8.163 shows how to access the type property. When the user clicks the Get Type button, an alert box appears indicating the type of input.

Listing 8.163 Accessing the FileUpload Type Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function informs the user what type of input the first element is.
function getType(){
    var mytype = document.form1.uploadbox.type;

    alert("The input box type is: " + mytype);
}
// -->
</script>

<form name="form1">
<input type="file" name="webfile">
<br><br>
Interested in finding out what type of input box is above?
Click on the button below.
<br>
<input type="button" value="Get Type" onClick='getType()'>
</form>
</body>
</html>
```

FileUpload.unwatch()

JavaScript 1.2+

Nav4+

Syntax

```
fileupload.unwatch(prop)
```

Description

The `unwatch()` method of the `FileUpload` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, *prop*, which is the property to unwatch.

Example

Listing 8.164 shows how the `unwatch()` method is used to disable the watchpoint set on the `name` property.

Listing 8.164 Using the `unwatch()` Method of the `FileUpload` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " </b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form1.uploadbox.name;
    watch("myProp",alertme);
    myProp = "yahoo";
    unwatch("myProp");
    myProp = "ebay";
}

// -->
</script>
<body>

<form name="form1">
Please select a file.
<input type="file" name="uploadbox">
<br><br>
Click on the button to change the FileUpload name property
<br>
```

```

<input type="button" value="Click Here" onClick='setWatch() '>
</form>

</body>
</html>

```

FileUpload.value

JavaScript 1.0+
Nav2+, Opera3+

Syntax

fileupload.value

Description

The value property of the FileUpload object specifies either the filename of the file selected or input by the user.

Example

Listing 8.165 shows how to access the value property. The form object is used in conjunction with the upload box name to get the value attribute of the FileUpload box.

Listing 8.165 *Accessing the FileUpload Value Property*

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function showFile(){
    var input = document.form1.uploadbox.value;
    alert("The filename entered is: " + input);
}
// -->
</script>

<form name="form1">
Please select a file.
<input type="file" name="uploadbox">
<br><br>
Click on the button to see the value of the FileUpload object.
<br>
<input type="button" value="Submit" onClick=showFile()>
</form>

</body>
</html>

```

FileUpload.watch()

JavaScript 1.2+

Nav4+

Syntax

```
fileupload.watch(prop, handler)
```

Description

The `watch()` method of the `FileUpload` object is used to watch for changes to `FileUpload` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.166 shows how the `watch()` method is used to set a watchpoint on the `name` property. When the upload box name is changed, the `alertme` function is called to display the change to the user.

Listing 8.166 Using the `watch()` Method of the `FileUpload` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    ➤ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form1.uploadbox.name;
    watch("myProp",alertme);
    myProp = "yahoo";
    myProp = "ebay";
}

// -->
</script>
<body>

<form name="form1">
Please select a file.
<input type="file" name="uploadbox">
<br><br>
Click on the button to change the FileUpload name property
<br>
```

```

<input type="button" value="Click Here" onClick='setWatch()'>
</form>

</body>
</html>

```

Form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Form object represents an HTML property created by the <form> tag. The Form object can be used to access all the properties of the specified form. Forms can be referenced either by the forms array or directly by name. Table 8.14 shows the different Form methods and properties.

Table 8.14 Properties and Methods of the Form Object

Type	Item	Description
Property	action	HTML action attribute of the Form object
	elements	Array reflecting elements within a form
	elements.length	Length of the elements array
	encoding	HTML enctype attribute of the Form object
	length	Number of elements within a form
	method	HTML method attribute of the Form object
	name	HTML name attribute of the Form object
	onReset	Event handler for the Reset button
	onSubmit	Event handler for the Submit button
	target	HTML target attribute of the Form object
Method	handleEvent()	Handles a specific event
	reset()	Resets form elements
	submit()	Submit for data
	unwatch()	Removes a watchpoint on a Form property
	watch()	Sets a watchpoint on a Form property

Example

Listing 8.167 uses the method of accessing the Form object directly by name.

Listing 8.167 Accessing the `Form` Object by Name

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function shows the name entered by the user to verify that
// the information is correct.
function checkName(){

    var firstName = document.formEx.first.value;
    var lastName = document.formEx.last.value;
    alert("The name you entered is: " + firstName + " " + lastName);
}
// -->
</script>

<form name="formEx">
First Name:
<input type="text" name="first" size="20">
Last Name:
<input type="text" name="last" size="25">
<br><br>
Click the button to check that the information is correct.
<br>
<input type="button" value="verify" name="check" onClick='checkName()'>
</form>
</body>
</html>
```

Listing 8.168 accesses the `Form` object by using the `forms` array. The form simply contains one text box. Clicking the button shows an alert box containing the name of the form. Because it is the first form in the document, it is index 0 in the `forms` array.

Listing 8.168 Using the `forms` Array

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function showFormName(){
    var name = document.forms[0].name;

    alert("The name of the form is: " + name);
}
```



```
// -->
</script>

<form name="form1">
This text box belongs to a form.
<input type="text" name="street">
<br><br>
Click on the button to get the name of the form.
<br>
<input type="button" value="Get Form name" onClick='showFormName()'>
</form>

<form name="form2">
This is the second form in the document. It contains a FileUpload object.
<input type="file" name="uploadbox" size="25">
</form>
</body>
</html>
```

Form.action

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.action

Description

The action property represents the action attribute of the HTML <form> tag. It is typically the URL to which the form is being submitted.

Example

Listing 8.169 shows one method of how to use the Form object to access the action property of the HTML form. The action property specifies the server and program to which the form is submitted.

Listing 8.169 Accessing the Action Value of the Form Object

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function getAction(){
    var actionValue = document.form1.action;
    document.form1.msg.value =
        "Your form was submitted to the following URL:\n " + actionValue;
```

Listing 8.169 Continued

```
}  
// -->  
</script>  
  
<form name="form1" action="http://www.test.org/cgi-bin/process.pl">  
Enter your street address:  
<input type="text" name="address" size="40">  
<br><br>  
<input type="button" value="Submit" onClick='getAction()'>  
<br><br><br>  
<textarea name="msg" rows="5" cols="62"></textarea>  
</form>  
</body>  
</html>
```

Form.elements

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.elements

Description

The `elements` property of the `Form` object represents the `elements` array, which is used to access each element within a form. The order of the form elements in the `elements` array is the order in which they appear in the HTML source.

Example

Listing 8.170 shows how to access form elements using the `elements` array. Because it is accessing the first element in the form, the index 0 is used. The second element in the form can be accessed with `elements[1]`.

Listing 8.170 Accessing Form Elements

```
<html>  
  
<body>  
<script type="text/javascript" language="JavaScript">  
<!--  
function getName(){  
    var textName = document.form1.elements[0].name;  
    alert("The textbox name is: " + textName);  
}  
// -->  
</script>  
  
<form name="form1">
```

This is a blank input textbox. Click on the button below to get the name of the textbox.

```
<br>
<input type="text" name="textbox1" size=25>
<br><br>
<input type="button" value="Get Name" onClick = 'getName()'>
</form>
</body>
</html>
```

Form.elements.length

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.elements.length

Description

The `elements.length` property of the Form object specifies the number of items in the `elements` array. Each item in the array refers to an object in the HTML form.

Example

Listing 8.171 shows how to use the `elements.length` property. When the user clicks the Get Elements button, an alert box is displayed indicating the number of elements in the form.

Listing 8.171 Using the `elements.length` Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// function uses the elements.length property to get the number
// of elements in the form.
function getNum(){
    var numOfElements = document.form1.elements.length;
    alert("The number of elements in this document are:" + numOfElements);
}
// -->
</script>

<form name="form1">
Dummy text box:
<input type="text" name="textbox1" size="25">
<br>
```

Listing 8.171 Continued

```
<input type="button" value="Dummy Button">
<input type="text" size="20" name="Sample">
<br><br>
Click on the button to get the number of elements in this form.
<input type="button" value="Get Elements" onClick='getNum()'>
</form>
</body>
</html>
```

Form.encoding

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.encoding

Description

The encoding property of the Form object represents the type of encoding used by the form. It can be specified in the HTML <form> tag as the enctype attribute.



NOTE

Setting the encoding property will override the HTML enctype attribute.

Example

In Listing 8.172, the encoding property of the Form object is used to get the type of encoding being used by the form.

Listing 8.172 Accessing Form Encoding Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// This function returns the encoding type for the specified form.
function getEncoding(){
    var encodingType = document.form1.encoding;
    return encodingType;
}
// -->
</script>

<form name="form1" action="post" enctype="application/x-www-form-urlencoded">
```

```
<input type="button" value="Get Encoding" onClick='getEncoding()'>
</form>
</body>
</html>
```

Form.handleEvent()

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

form.handleEvent(event)

Description

The `handleEvent()` method of the `Form` property invokes the handler for the specified event. It takes one parameter, which is the event to be handled.

Example

In Listing 8.173, the `handleEvent()` method is being used to handle the event being passed. The script captures the `CLICK` event. When this occurs, the `handleMyClick()` function calls the `handleEvent()` method to handle the `CLICK` event.

Listing 8.173 *Using the* `handleEvent()` *Method*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

// Sets up the document to capture CLICK events.
document.captureEvents(Event.CLICK);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handleMyClick(evnt){
    window.document.button1.handleEvent(evnt);
}

function displayMsg(){
    document.form1.msg.value += "Click event occurred.\n";
}

// This registers the handle function as the event handler for the
// CLICK event.
document.onClick = handleMyClick;
```

Listing 8.173 Continued

```
// -->
</script>

<form name="form1">
<input type="button" value="Click on Me" name="button1" onClick='displayMsg()'>
<br><br>
<textarea name="msg" rows="10" cols="50" onClick='displayMsg()'></textarea>
</form>

</body>
</html>
```

Form.length

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.length

Description

The length property of the Form object represents the number of elements within a form.



NOTE

This property works the same as the `elements.length` property.

Example

Listing 8.174 shows an example of how the length property is used to determine the number of elements in the document. The `showNumElements()` function informs the user of the form length, which represents the number of form elements.

Listing 8.174 Using the Form.length Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// function displays an alert box indicating the number of elements
// in the form.
function showNumElements(){
    alert("There are " + document.form1.length +
    ➤ " elements in this document");
}
```

```
// -->
</script>

<form name="form1">
Enter First Name:
<input type="text" size="15"><br>
Enter Last Name:
<input type="text" size="20"><br>
Enter address:
<input type="text" size="40"><br><br>
<input type="button" value="Submit" onClick='showNumElements()' '>
</form>

</body>
</html>
```

Form.method

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.method

Description

The method property of the Form object represents the type of submission, GET or POST, being used by the form.

Example

In Listing 8.175, the method property is used to get the type of method being used by the form. The `informMethod()` function alerts the user of the method, GET or POST, being used by the form.

Listing 8.175 Accessing the method Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

function informMethod(){
alert("The form method is:" + document.form1.method);
}
// -->
</script>
```

Listing 8.175 Continued

```
<form name="form1" method="get">

First Name:<input type="text" name="first" size="15">
Last Name:<input type="text" name="last" size="25">
<br>
City:<input type="text" name="city" size="20">
State:<input type="text" name="state" size="2" maxlength="2">
Zip:<input type="text" name="zip" size="5" maxlength="5">
<br><br>
Click the button to see what type of Method is used for submission.
<input type="button" value="Click Here" onClick='informMethod()'>
</form>
</body>
</html>
```

Form.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.name

Description

The name property of the Form object represents the name of the form as specified in the HTML <form> tag.

Example

Listing 8.176 shows how the name property is used to get the HTML name attribute of the form. The showName function uses the Form object to access the name attribute of the form.

Listing 8.176 Accessing the name Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function showName(){
// Alert box tells what the name of the form is
alert("Form Name is: " + document.form1.name);
}
// -->
</script>

<form name ="form1" >
```


Dummy input text box.

```
<input type= "text" size="15">
<br><br>
Click on the button to get the name of the form
<input type="button" value="click me" onclick='showName()'>

</form>
</body>
</html>
```

Form.onReset

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
onReset="command"
```

Description

The onReset method of the Form object executes JavaScript code when a reset event occurs.

Example

Listing 8.177 demonstrates the use of the onReset event handler. The JavaScript function checks to see if the Reset button has been clicked. If so, all the text values are reset to a specified value.

Listing 8.177 Using the onReset Event Handler

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function displays a message when the onReset event handler determines that
// the reset button has been pressed.
function showMsg(){
    document.form2.msg.value = "You have cleared the Entry Fields.";
}
// -->
</script>

<form name="form1" onReset='showMsg()'>
Entry 1:<input type= "text" name="text1" sizesize="20"><br>
Entry 2:<input type= "text" name="text2" sizesize="20"><br>
Entry 3:<input type= "text" name="text3" sizesize="20"><br>
<input type="reset" value="reset" name="Reset"
```

Listing 8.177 Continued

```
</form>

<form name="form2">
Message:<input type="text" name="msg" size="50">
<br><br>
</form>
</body>
</html>
```

Form.onSubmit

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onSubmit="command"
```

Description

The onSubmit method of the Form property executes JavaScript code when a submit event occurs.

Example

Listing 8.178 shows how the onSubmit event handler is used for a Form object. A user would enter her comments in the text area provided. When finished, she clicks the Submit button to submit the form. Using the onSubmit event handler, you can display a thank you note when the user has submitted her comments.

Listing 8.178 Using the onSubmit() Form Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
//Function displays a confirmation message when a form is submitted.
function confirm(){
    alert("Your comments have been submitted. Thank you.");
}
// -->
</script>

<form name="form1" onSubmit = 'confirm()'  
<b>Enter Comments:</b>
<br>
<textarea name="comments" rows="5" cols="60"></textarea>
<br><br>
```

```
<input type = "submit" Value="Submit Comments">
</form>
</body>
</html>
```

Form.reset()

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
form.reset()
```

Description

The reset method of the Form object resets all the form elements to their default values. It operates the same as a mouse click on a Reset button for the calling form.

Example

Listing 8.179 shows how the reset method is used to reset a form. When the Reset button is clicked, all values in the form object are reset (text boxes are cleared of their values).

Listing 8.179 Accessing the reset() Method

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function resetForm(form){
    document.form1.reset(form);
}
// -->
</script>

<form name="form1">
Field 1:<input type="text" size="20" name="text1"><br>
Field 2:<input type="text" size="20" name="text2"><br>
Field 3:<input type="text" size="20" name="text3"><br>
<input type="button" name="reset" value="reset" onClick='resetForm(this.form)'\>
</form>

</body>
</html>
```

Form.submit()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
form.submit()
```

Description

The `submit()` method of the `Form` object is used to submit a form. It operates the same as if a Submit button was clicked.

Example

Listing 8.180 shows how you would submit a form using the `submit()` method. The form is submitted to the value specified in the `action` attribute of the HTML `<form>` tag. In this specific example, a made up script processes the form.

Listing 8.180 *Accessing the `submit()` Method*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// Function submits a form to a server specified in the <form> tag
function submitForm(form){
    document.form1.submit(form);
}
// -->
</script>

<form name= "form1" method="post"
➤ action="http://www.myserver.com/cgi-bin/test.pl">
This is a sample form
<br><br>
Name:<input type="text" size="40" name="name">
<br>
Age:<input type="text" size="3" name="age">
<br>
Phone Number:<input type="text" size="10" name="phone">
<br><br>
<input type= "button" value="Submit" onclick = 'submitForm(this.form)''>
</form>
</body>
</html>
```

Form.target

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

form.target

Description

The target property of the Form object represents the target window or frame in which the form results will be displayed. This can also reflect the target attribute of the HTML <form> tag. Valid target attributes are: _blank, _parent, _self, and _top.

Example

In Listing 8.181, a sample form is created for submission. When the Submit button is clicked, the show() function is called to inform the user whether the target property has been specified or not.

Listing 8.181 *Displaying Results Using the target Property*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
function show(){

var tar = document.form1.target;

    if(tar == "_self"){
        alert("The target for this form submission is this window");
    }
    else{
        alert("The target has not been specifically specified");
    }
}
// -->
</script>

<form name="form1" action="post" target="_self">
First Name:<input type="text" size="15" name="first">
Last Name:<input type="text" size="20" name="last">
<br>
Street:<input type="text" size="40" name="street">
<br>
City:<input type="text" size="15" name="city">
State:<input type="text" size="2" name="st">
Zip:<input type="text" size="5" name="zip">
```

Listing 8.181 Continued

```
<br><br><br>
<input type="button" value="submit" onClick='show()'>
</form>

</body>
</html>
```

Form.unwatch()

JavaScript 1.2+
Nav4+

Syntax

form.unwatch(prop)

Description

The `unwatch()` method of the `Form` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, *prop*, which is the property to unwatch.

Example

Listing 8.182 shows how the `unwatch()` method is used to disable the watchpoint on the form name.

Listing 8.182 Using the `unwatch()` Method of the `Form` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + 
    "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.forms[0].name;
    watch("myProp",alertme);
    myProp = "red_form";
    unwatch("myProp");
    myProp = "blue_form";
}

// -->
</script>
<body>
```

```

<form name="form1">
<input type="button" value="Click Here" onClick='setWatch()'>
</form>

</body>
</html>

```

Form.watch()

JavaScript 1.2+
Nav4+

Syntax

form.watch(prop, handler)

Description

The `watch()` method of the `Form` object is used to watch for changes to `Form` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user defined function.

Example

Listing 8.183 shows how the `watch()` method is set on the form name property. When the button is clicked, the watchpoint is set and then the property is changed. Each time it is changed, the `alertme` function is called to display the changes.

Listing 8.183 Using the `watch()` Method of the `Form` Object

```

<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
" </b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.forms[0].name;
    watch("myProp",alertme);
    myProp = "red_form";
    myProp = "blue_form";
}

// -->
</script>
<body>

```

Listing 8.183 Continued

```
<form name="form1">
<input type="button" value="Click Here" onClick='setWatch()'>
</form>

</body>
</html>
```

Frame

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

A window can display multiple, independently scrollable frames on a single screen, each with its own distinct URL. The `Frame` object, which has an instance created with each occurrence of the `<frame>` tag, is a convenience for thinking about the objects that make up these frames. However, JavaScript actually represents a frame using a `Window` object. Every `Frame` object is a `Window` object, and has all the methods and properties of a `Window` object. Table 8.15 shows the available methods and properties of `Frames`.

NOTE

The majority of examples for the `Frame` object assume that `top.html` and `bottom.html` files exist. Some examples also only display the JavaScript code that would be contained in a document.

Table 8.15 Properties and Methods of the `Frame` Object

Type	Item	Description
Property	<code>document</code>	Current document loaded within the frame
	<code>frames</code>	Array containing references to the child frames
	<code>length</code>	Length of the frames array
	<code>name</code>	HTML <code>name</code> attribute of the <code>Frame</code> object
	<code>onBlur</code>	Event handler for the <code>blur</code> event
	<code>onFocus</code>	Event handler for the <code>focus</code> event
	<code>onMove</code>	Event handler for the <code>move</code> event
	<code>onResize</code>	Event handler for the <code>resize</code> event
	<code>parent</code>	Main window or frame from which child frames are created
	<code>self</code>	Refers to the current frame
	<code>top</code>	Browser window that executes script
	<code>window</code>	Refers to the current window or frame

Type	Item	Description
Method	<code>blur()</code>	Removes focus from the frame
	<code>clearInterval()</code>	Cancels a repeated execution
	<code>clearTimeout()</code>	Cancels any delayed execution
	<code>focus()</code>	Applies focus to the frame
	<code>print()</code>	Invokes the print dialog box
	<code>setInterval()</code>	Sets a function schedule for repeated execution
	<code>setTimeout()</code>	Sets a function schedule for delayed execution
	<code>unwatch()</code>	Removes a watchpoint on a Frame property
	<code>watch()</code>	Sets a watchpoint on a Frame property

Example

Listing 8.184 shows the code to create an HTML frame and access its properties. This example assumes that the files `top.html` and `bottom.html` already exist.

Listing 8.184 Frame *Example*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Function returns the name of the frame
function getName(){
    var frameName = document.window.frame.name;
    return frameName;
}
// -->
</script>

<frameset rows="100,*">
<frame src="top.html" name="upper">
<frame src="bottom.html" name="bottom" scrolling="yes">
</frameset>

</html>
```

Frame.blur()

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
frame.blur()
```

Description

The `blur()` method of the `Frame` object removes focus from the frame.

Example

Listing 8.185 uses the `blur()` method to remove focus from the frame.

Listing 8.185 *Example of the `blur()` Method*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
function removeFocus(){
    document.upper.blur()
}
// -->
</script>

<frameset rows="100,*">
<frame src="top.html" name="upper" onDblick='removeFocus() '>
<frame src="bottom.html" name="bottom" scrolling="yes">
</frameset>

</html>
```

Frame.clearInterval()

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

```
frame.clearInterval()
```

Description

The `clearInterval()` method of the `Frame` object is used to cancel a repeated execution.

Example

Listing 8.186 shows the syntax for accessing the `clearInterval()` method. For a more detailed example, see `window.clearInterval()`.

Listing 8.186 *Example of the `clearInterval()` Method*

```
<script type="text/javascript" language="JavaScript">
<!--
function cancel(){
    framename.clearInterval();
```

```
}  
// -->  
</script>
```

Frame.clearTimeout()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
frame.clearTimeout()
```

Description

The `clearTimeout()` method of the `Frame` object is used to cancel a delayed execution. For a more detailed example, see `window.clearTimeout()`.

Example

Listing 8.187 shows the syntax for accessing the `clearTimeout()` method.

Listing 8.187 Accessing the `clearTimeout()` Method

```
<script type="text/javascript" language="JavaScript">  
<!--  
function stop(){  
    frameName.clearTimeout();  
}  
// -->  
</script>
```

Frame.document

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
frame.document
```

Description

The `document` property of the `Frame` object contains information about the current document. The `document` property is created by the HTML `<body>` tag and is available in every `Frame` or `Window` object. The `document` property is used to access other aspects of the HTML document.

Example

Listing 8.188 shows how the `document` property can be used to access document elements within a specific frame. For a more detailed example, see `window.document`.

Listing 8.188 Accessing the document *Property*

```
<script type="text/javascript" language="JavaScript">
<!--
function getName(){
    var name = frameName.document.elements[1].name;
}
// -->
</script>
```

Frame.focus()

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
frame.focus()
```

Description

The `focus()` method of the `Frame` object is used to set the focus to a specific frame.

Example

Listing 8.189 shows the syntax for using the `focus()` method. When the document is loaded, the focus is automatically set to the bottom frame.

Listing 8.189 Accessing the `focus()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
//Set initial focus to bottom frame.
document.bottom.focus;
// -->
</script>

<frameset rows="80, *">
<frame name=top src=top.html>
<frame name=bottom src=bottom.html>
</frameset>

</html>
```

Frame.frames

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`frame.frames[num]`

Description

The `frames` property of the `Frame` object represents an array that stores child frame objects. Array entries of the child frame can be referenced either by index number or by the name assigned from the `name` attribute.

Example

In Listing 8.190, the `frames` array is used to access the length of the child `frame`.

Listing 8.190 Example Using the `frames` Property

```
<script type="text/javascript" language="JavaScript">
<!--
function getFrameLength(){
    var childLength = document.frames["firstChild"].length;
}
// -->
</script>
```

Frame.length

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`frame.length`

Description

The `length` property of the `frames` object represents the length of the `frames` array, which is the number of child frames.

Example

Listing 8.191 shows a simple function call that uses the `length` property to set a variable for the number of child frames.

Listing 8.191 Example Using the `length` Property

```
<script type="text/javascript" language="JavaScript">
<!--
function getLength(){
    var numOfChildFrames = window.frameName.length;
}
// -->
</script>
```

Frame.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

frame.name

Description

The `name` property of the `Frame` object represents the name given to a frame as specified from the `name` attribute of the HTML `<frame>` tag.

Example

In Listing 8.192, the `name` property is used to inform the user of the active frame.

Listing 8.192 *Example of the `name` Property*

```
<script type="text/javascript" language="JavaScript">
<!--
function activeFrame(){
    var frameName = window.myframe.name;
    alert("The active frame is: " + frameName);
}
// -->
</script>
```

Frame.onBlur

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`onBlur="command"`

Description

The `onBlur` event handler specifies what happens when the focus has been removed from a frame.

Example

In Listing 8.193, the `onBlur` event handler is used to change the color of the upper frame when the focus is set on it.

Listing 8.193 *Using the `onBlur` Event Handler*

```
<html>
```

```
<script type="text/javascript" language="JavaScript">
```

```

<!--
function change(){
    frames[0].onblur=new Function("document.bgColor='green'")
}
// -->
</script>

<frameset rows="50%,*" onload='change()'>
<frame name="topFrame" src="top.html">
<frame name="bottomFrame" src="bottom.html">
</frameset>

</html>

```

Frame.onFocus

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

onFocus="command"

Description

The onFocus event handler is used to specify when the focus is brought to a frame. The user can set the focus either by pressing the mouse button or by using the Tab key.

Example

The syntax for accessing the onFocus event handler is shown in Listing 8.194. When the focus is set on the top frame, an alert box is displayed.

Listing 8.194 Syntax for the onFocus Event Handler

```

<html>

<frameset rows="80, *">
<frame src="top.html" name="top" onFocus='alert("You are now in the top
frame")'>
<frame src="bottom.html" name="bottom">
</frameset>

</html>

```

Frame.onMove

JavaScript 1.2+, ECMAScript 1.0+, JScript 1.0+
Nav4+, IE3+

Syntax

`onMove="command"`

Description

The `onMove` frame event handler is used to specify what happens when a move event occurs within a frame.

Example

Listing 8.195 shows the `onMove` event handler being used to inform the user when the frame is being moved.

Listing 8.195 *Example of the `onMove` Event Handler*

```
<html>

<frameset rows="80, *" onMove='alert("You are now moving the frame")'>
<frame src="top.html" name="top" >
<frame src="bottom.html" name="bottom">
</frameset>

</html>
```

Frame.onResize

JavaScript 1.2+, ECMAScript 1.0+
Nav4+

Syntax

`onResize="command"`

Description

The `onResize` frame event handler is used to specify when a frame has been resized.

Example

Listing 8.196 shows how you would use the `onResize` event handler to check for when a frame has been resized.

Listing 8.196 *Syntax of the `onResize` Event Handler*

```
<html>

<frameset rows="80, *">
```



```
<frame src="top.html" name="top" >
<frame src="bottom.html" name="bottom"
  ⚡ onResize='alert("You are resizing the bottom frame")'>
</frameset>

</html>
```

Frame.parent

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

frame.parent

Description

The `parent` property of the `Frame` object specifies the frame containing the current frame.

Example

Listing 8.197 shows a small example of how the name of the `Frame` parent can be accessed.

Listing 8.197 *Example of the* `Frame.parent` *Property*

```
<script type="text/javascript" language="JavaScript">
<!--
function getParent(){
    var parentName = parent.frameName.name;
}
// -->
</script>
```

Frame.print()

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

frame.print(options)

Description

The `print` method of the `Frame` object is used to send the document output of a particular frame to a printer. This works the same as executing the `Print` command from a browser menu.

Example

Listing 8.198 shows an example of the `print` method. Using the `onClick` event handler, the `print()` method is executed when the bottom frame is clicked.

Listing 8.198 *Example of the `print()` Method*

```
<html>

<frameset rows="80, *" >
<frame src="top.html" name="top" >
<frame src="bottom.html" name="bottom" onClick='document.window.bottom.
print()'>
</frameset>

</html>
```

Frame.self

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`frame.self`

Description

The `self` property of the `Frame` object is used as a keyword to reference the current frame.

Example

Listing 8.199 shows an example of the `self` property being used to get the name for the current frame.

Listing 8.199 *Syntax for the `self` Property*

```
<script type="text/javascript" language="JavaScript">
<!--
function getName(){
    var name=window.frame.self.name;
}
// -->
</script>
```

Frame.setInterval()

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

```
frame.setInterval(exp,num)
```

Description

The `setInterval()` method of the `Frame` object is used to schedule a function for repeated execution. It takes two parameters. The first parameter, *exp*, is the expression being executed. The second parameter, *num*, is the time in milliseconds that elapses before the expression is executed again. The function being executed is stopped by calling the `clearInterval()` method.

Example

Listing 8.200 shows a simple example of a `setInterval()` method call. For a more detailed example, see `window.setInterval`.

Listing 8.200 Example of the `setInterval()` Method Call

```
<script type="text/javascript" language="JavaScript">
<!--
Frame.setInterval(document.write('Begin writing', 50);
// -->
</script>
```

Frame.setTimeout()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
frame.setTimeout(str,num)
```

Description

The `setTimeout()` method of the `Frame` object is used to schedule a function for delayed execution. The method accepts two parameters. The first parameter, *str*, is the function or expression being evaluated. The second parameter, *num*, is a numeric value specifying the millisecond units that elapse before the function or expression is executed.

Example

Listing 8.201 shows an example of the `setTimeout()` method being used to delay the printing of `text`.

Listing 8.201 Example of the `setTimeout()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
```

Listing 8.201 Continued

```
document.bottom.setTimeout(document.bottom.print("Keep on running"), 500);
// -->
</script>

<frameset rows="100,*">
<frame src="top.html" name="upper">
<frame src="bottom.html" name="bottom" scrolling="yes">
</frameset>

</html>
```

Frame.top

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

frame.top

Description

The `top` property of the `Frame` object specifies the top-most browser window containing frames.

Example

Listing 8.202 shows an example of how the `top` property is used to get the name of the `top` frame.

Listing 8.202 Example of the `top` Property

```
<script type="text/javascript" language="JavaScript">
<!--

var topBrowserName = myframe.document.top.name;
alert("The name of the topmost browser is: " + topBrowser);
// -->
</script>
```

Frame.unwatch()

JavaScript 1.2+
Nav4+

Syntax

frame.unwatch(prop)

Description

The `unwatch()` method of the `Frame` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, *prop*, which is the property to unwatch.

Example

Listing 8.203 shows how the `unwatch()` method is used to remove the watchpoint set on the frame name. The `setWatch` method sets the watchpoint on the frame name initially. When the name is changed, the `alertme` function is called. The watchpoint is then removed using the `unwatch()` method.

Listing 8.203 Using the `unwatch()` Method of the `Frame` Object

```
<script type="text/javascript" language="JavaScript">
<!--

// Alert user when watch property changed.
function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " " + "</b>" + " to " + " " + "<b>" + newValue + "</b><br>")
    return newValue;
}

// Set the watchpoint on the frame name.
function setWatch(){
    myProp = document.frameName.name;
    watch("myProp",alertme);
    myProp = "bugs";
    unwatch("myProp");
    myProp = "daffy";
}

// -->
</script>
```

Frame.watch()

JavaScript 1.2+
Nav4+

Syntax

frame.watch(prop, handler)

Description

The `watch()` method of the `Frame` object is used to watch for changes to `Frame` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.204 shows how the `watch()` method is used to set a watchpoint on the frame name. The `setWatch` method sets the watchpoint on the frame name initially. When the name is changed, the `alertme` function is called.

Listing 8.204 Using the `watch()` Method

```
<script type="text/javascript" language="JavaScript">
<!--

// Alert user when watch property has been changed.
function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    ➤ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

// Set the watchpoint on the frame name.
function setWatch(){
    myProp = document.frameName.name;
    watch("myProp",alertme);
    myProp = "bugs";
}

// -->
</script>
```

Frame.window

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`frame.window`

Description

The `window` property of the `Frame` object is used to reference the current frame. This works the same as using the `self` property.

Example

Listing 8.205 shows an example of the syntax of the `window` property.

Listing 8.205 Syntax for the `window` Property

```
<html>

<script type="text/javascript" language="JavaScript">
```

```

<!--
//Sets focus to the upper frame.
document.upper.window.focus();
// -->
</script>

<frameset rows="100,*">
<frame src="top.html" name="upper">
<frame src="bottom.html" name="bottom">
</frameset>

</html>

```

Hidden

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Hidden object is created using the HTML `<input>` tag. Specifying the `type` parameter of the `<input>` tag as `hidden` creates the Hidden object. It is a text object that is not visible in an HTML form. The Hidden object is primarily used for passing name/value pairs from a form. Table 8.16 shows the properties of the Hidden object.

Table 8.16 *Properties and Methods of the Hidden Object*

Type	Item	Description
Property	form	Specifies the form containing the Hidden object.
	name	Refers to the name of the Hidden object.
	type	Refers to the HTML <code>type</code> attribute of the Hidden object.
	value	Refers to the HTML <code>value</code> attribute of the Hidden object.
Method	unwatch()	Removes a watchpoint on a Hidden property.
	watch()	Sets a watchpoint on a Hidden property.

Example

Listing 8.206 shows how a Hidden object is created and how some of its properties are accessed.

Listing 8.206 Creating a Hidden Object

```
<html>

<body>
<form name="form1">
<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Get Hidden Attributes"
  onClick='alert("The Hidden object Name is: " + form1.hide1.name +
➤ " The Hidden Type is: " + form1.hide1.type + " The Hidden Value is: "
➤ + form1.hide1.value)'\>
</form>

</body>
</html>
```

Hidden.form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

hidden.form

Description

The form property of the Hidden object is used to reference the form containing the Hidden object.

Example

Listing 8.207 shows how the form property is used to store the name of the form.

Listing 8.207 Example of the form Property

```
<html>

<body>
<form name="form1">
Form name:<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Form Name"
  onClick="this.form.hide1.value=this.form.name">
</form>

</body>
</html>
```

Hidden.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`hidden.name`

Description

The name property of the Hidden object is used to get the name of the Hidden object. This is the HTML name attribute for the Hidden object.

Example

Listing 8.208 shows an example of how the name property is used. The form object is used to access the name property.

Listing 8.208 Example of the name Property

```
<html>

<body>
<form name="form1">
Form name:<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Hidden Name"
    onClick='alert("The Hidden object name is: " + form1.hide1.name)'\>
</form>

</body>
</html>
```

Hidden.type

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

`hidden.type`

Description

The type property of the Hidden object specifies the hidden type. For all Hidden objects, the type value is Hidden.

Example

Listing 8.209 shows an example of how the type property is used to get the hidden type. The form object is used to access the type property.

Listing 8.209 Example of the `type` *Property*

```
<html>

<body>
<form name="form1">
Form name:<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Hidden Type"
    onClick='alert("The Hidden object type is: " + form1.hide1.type)'\>
</form>

</body>
</html>
```

Hidden.unwatch()

JavaScript 1.2+
Nav4+

Syntax

hidden.unwatch(prop)

Description

The `unwatch()` method of the `Hidden` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to `unwatch`.

Example

Listing 8.210 shows how the `unwatch()` method is used to turn off the watch point set on the `Hidden` object value.

Listing 8.210 Using the `unwatch()` Method of the `Hidden` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " " + "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form1.hide1.value;
    watch("myProp",alertme);
    myProp = "joe";
}
```

```

    unwatch("myProp");
    myProp = "jane";
}

// -->
</script>
<body>
<form name="form1">
Hidden Input<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Click Here" onClick='setWatch()'>
</form>

</body>
</html>

```

Hidden.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

hidden.value

Description

The value property of the hidden object reflects the HTML value attribute of the Hidden object.

Example

Listing 8.211 shows how the value property of the Hidden object is used. The onClick event handler is used to display an alert box to the user indicating the value of the Hidden object.

Listing 8.211 *Using the value Property of the Hidden Object*

```

<html>

<body>
<form name="form1">
Form name:<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Get Hidden Value"
    onClick='alert("The Hidden object value is: " + form1.hide1.value)'\>
</form>

</body>
</html>

```

Hidden.watch()

JavaScript 1.2+
Nav4+

Syntax

hidden.watch(prop, handler)

Description

The `watch()` method of the `Hidden` object is used to watch for changes to `Hidden` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.212 shows how the `watch()` method is used to check for assignments to the `hidden` object value. If an assignment is made, the `alertme` function is called to notify the user.

Listing 8.212 Using the `watch()` Method of the `Hidden` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + "</b>"
    + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form1.hide1.value;
    watch("myProp",alertme);
    myProp = "joe";
    myProp = "jane";
}

// -->
</script>
<body>
<form name="form1">
Hidden Input<input type="hidden" name="hide1" value="Test">
<p>
<input type="button" value="Click Here" onClick='setWatch() '>
</form>

</body>
</html>
```

History

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The History object is a predefined JavaScript object that allows you to navigate through the history of Web sites that a browser has displayed. The browser stores a history of visited URLs in a list, which the History object references. Table 8.17 shows the methods and properties of the History object.

Table 8.17 *Properties and Methods of the History Object*

Type	Item	Description
Property	current	Refers to the current URL in the history list
	length	Returns the number of entries in the history list
	next	Refers to the next URL in the history list
	previous	Refers to the previous URL in the history list
Method	back()	Loads the URL for the previously visited Web site
	forward()	Loads the next URL in the history list
	go()	Loads a URL from the history list
	unwatch()	Removes a watchpoint on a History property
	watch()	Sets a watchpoint on a History property

Example

Listing 8.213 shows an example of how the History object is used. A for loop is used to loop through the history list and outputs the Web site entries in the list.

Listing 8.213 *Example of the History Object*

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
// code loops though the history list and outputs a history list
// of web sites visited.
for (i=0; i<history.length; i++){
document.writeln(window.history.previous);
}
//-- End Hide>
</script>

</body>
</html>
```

History.back()

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`history.back()`

Description

The `back()` method of the History object is used to load the URL for the previously visited Web site.

Example

Listing 8.214 shows an example of how a JavaScript button can use the `back()` method to simulate the browser's back functionality.

Listing 8.214 *Example of the* `back()` *Method*

```
<html>

<body>

<form name="form1">
Click on the button to go back to the previous page.
<input type="button" value="Go Back" onClick='window.history.back()''>
</form>

</body>
</html>
```

History.current

JavaScript 1.1+, ECMAScript 1.0+, JScript 3.0+
Nav3+, IE4+

Syntax

`history.current`

Description

The `current` property of the History object contains a string that specifies the complete URL of the current history entry.

NOTE

In Navigator 4 and higher, getting the current value requires the `UniversalBrowserRead` privilege.

Example

Listing 8.215 shows an example of how the current property is used. For this example, the UniversalBrowserRead privilege must be set. When this is done, the onClick event handler alerts the user of the History current property.

Listing 8.215 *Example of the* current *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// Request the Universal Browser Read privilege.
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");

// -->
</script>

<form name="form1">
<input type="button" value="Get Current"
  ⚡ onClick='alert(window.history.current)'>
</form>

</body>
</html>
```

History.forward()

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
history.forward()
```

Description

The forward() method of the History object is used to load the URL for the next Web site in the history list.

Example

Listing 8.216 shows an example of how the forward() method is used to simulate the forward functionality of the browser.

Listing 8.216 Example of the forward() Method

```
<html>

<body>

<form name="form1">
Click on the button to go to the forward browser page.
<input type="button" value="Go Forward" onClick='window.history.forward() '>
</form>

</body>
</html>
```

History.go()

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

history.go(*num*)

Description

The go() method of the History object loads a URL from the history list.

Example

Listing 8.217 shows how the go() method can be used to navigate to another Web site in the history list.

Listing 8.217 Example of the History go() Method

```
<html>

<body>

<form name="form1">
Click on the button to go back 2 pages.
<input type="button" value="Go" onClick='window.history.go(-2) '>
</form>

</body>
</html>
```

History.length

JavaScript 1.0+, ECMAScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

history.length

Description

The `length` property of the `History` object is used to get the number of URLs in the history list.

Example

Listing 8.218 shows an example of how the `length` property is used to determine how many URLs are in the history list.

Listing 8.218 *Example of the `length` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
//Write the number of elements in the history list.
var numOfURL = window.history.length;
document.write("The number of URL's in the history list is: " + numOfURL);
// -->
</script>

</body>
</html>
```

History.next

JavaScript 1.1+, ECMAScript 1.0+, JScript 3.0+
 Nav3+, IE4+

Syntax

`history.next`

Description

The `next` property of the `History` object is used to get the URL for the next entry in the history list.

NOTE

In Navigator 4 and later, getting the next value requires the `UniversalBrowserRead` privilege.

Example

Listing 8.219 shows an example of how the `next` property is used to get the next URL in the history list. If nothing is displayed for the `nextURL`, you are at the end of the history list.

Listing 8.219 Example of the *next* Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Request the Universal Browser Read privilege.
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");

//Get the next URL in the history list.
var nextURL = window.history.next;
document.write("The next URL in the history list is: " + nextURL);
// -- >
</script>

</body>
</html>
```

History.previous

JavaScript 1.1+, ECMAScript 1.0+, JScript 3.0+
Nav3+, IE4+

Syntax

history.previous

Description

The previous property of the History object is used to get the URL for the previous entry in the history list.



NOTE

In Navigator 4 and later, getting the previous value requires the UniversalBrowserRead privilege.

Example

Listing 8.220 shows an example of how the previous property is used to get the previous URL in the history list.

Listing 8.220 Example of the *previous* Property

```
<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--
```

```
// Request the Universal Browser Read privilege.
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");
// -->
</script>

<form name="form1">
<input type="button" value="Get Previous"
➡ onClick='alert(window.history.previous)'\>
</form>

</body>
</html>
```

History.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

history.unwatch(prop)

Description

The `unwatch()` method of the `History` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to `unwatch`.

Example

Listing 8.221 shows how the `unwatch()` method is used to remove the watchpoint set on the `History` object.

Listing 8.221 Using the `unwatch()` Method on the `History` Object

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
➡ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myRes = window.history.previous;
    watch("myRes",alertme);
```

Listing 8.221 Continued

```
myRes = "something";
unwatch("myRes");
myRes = "otherthing";
}

// -->
</script>

<body>
<form name="form1">
<input type="button" value="Get Previous" onClick='setWatch()'>
</form>
</body>
</html>
```

History.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

history.watch(prop, handler)

Description

The `watch()` method of the `History` object is used to watch for changes to `History` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.222 shows how the `watch()` method is used to watch for changes to the `history` property.

Listing 8.222 Using the `watch()` Method on the `History` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " " + "<b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myRes = window.history.previous;
    watch("myRes",alertme);
}
```

```

    myRes = "something";
    myRes = "otherthing";
}

// -->
</script>

<body>

<form name="form1">
<input type="button" value="Get Previous" onClick='setWatch()'>
</form></body>
</html>

```

Image

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Image object represents an image that was created with the tag. Images can be downloaded and cached dynamically by using the Image() constructor, but they cannot be displayed using the constructor. The constructor takes two optional arguments, width and height. The argument *width* specifies the width of the image in pixels, whereas the argument *height* specifies the height of the image in pixels. If these arguments are larger than the actual image, the image will be stretched to these dimensions. You have probably noticed that there is no argument specifying the image to load. The image to load is specified using dot notation and the src property after the image constructor has been called. Table 8.18 lists all the properties, methods, and events associated with Image object.

Table 8.18 *Properties, Methods, and Events Associated with the Image Object*

Type	Item	Description
Property	border	Width of the border around an image.
	complete	Has the image finished loading?
	height	Height of the image.
	hspace	Padding on left and right of the image.
	lowsrc	Alternate image for low-resolution displays.
	name	Name of the image.
	src	URL of the image.

Table 8.18 Continued

Type	Item	Description
Method	<code>vspace</code>	Padding on top and bottom of the image.
	<code>width</code>	Width of the image.
	<code>handleEvent()</code>	Invokes an images event handler.
Event	<code>unwatch()</code>	Removes a watchpoint on an <code>Image</code> property.
	<code>watch()</code>	Sets a watchpoint on an <code>Image</code> property.
	<code>onAbort</code>	Handler when the image load is aborted.
	<code>onError</code>	Handler when an error occurs while loading the image.
	<code>onKeyDown</code>	Handler for <code>KeyDown</code> events within the image.
	<code>onKeyPress</code>	Handler for <code>KeyPress</code> events within the image.
	<code>onKeyUp</code>	Handler for <code>KeyUp</code> events within the image.
	<code>onLoad</code>	Handler when the image is finished loading.

Example

Listing 8.223 demonstrates how to use the `<img>` tag and the `Image()` constructor to alternate images in a document.

Listing 8.223 Example of the `Image` Object

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Alternate flag
alternate=0;

//Create an Image object and preload image.
circle = new Image();
circle.src = "circle.gif";
square = new Image();
square.src = "square.gif";

function changeImage()
{
    if(alternate==0)
    {
        document.magic.src=circle.src;
        alternate=1;
    }
    else
    {
        document.magic.src=square.src;
```

```
        alternate=0;
    }
}

// -->
</script>

<center>
<h2>Magic Trick</h2>

<form>
<input type="button"
        value="Change Image"
        onClick="changeImage()">
</form>


</center>

</html>
```

Image.border

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.border

Description

The border property of the Image object specifies the width of the border around an image in pixels. This property can only be set by the border attribute of the tag.

Example

Listing 8.224 shows how the border property is used.

Listing 8.224 *Example of the* border *Property*

```
<html>


<br>

<script type="text/javascript" language="JavaScript">
<!--
```

Listing 8.224 Continued

```
//Display width of border.
document.write("The image has a border width of ");
document.write(document.circle.border," pixels.");

// -->
</script>

</html>
```

Image.complete

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.complete

Description

The complete property of the Image object is a boolean value that specifies if an image has finished loading. After an image has completely loaded, the property is changed to false. If the load is aborted or an error occurs during the loading process, the property will be set to true.

Example

Listing 8.225 displays a message based on the value of the complete property.

Listing 8.225 Example of the complete Property

```
<html>


<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display message about loading progress of image.
if(document.circle.complete == true)
    document.write("The image has finished loading.");
else
    document.write("The image has not finished loading.");

// -->
</script>

</html>
```

Image.handleEvent()

JavaScript 1.2+,
Nav4+

Syntax

```
image.handleEvent(event)
```

Description

The `handleEvent()` method provides a way to invoke an image's event handler, even though the event never happened. The *event* argument associated with this method can be any of the events handled by the `Image` object.

Example

Listing 8.226 shows how to force an image to handle a `KEYDOWN` event.

Listing 8.226 *The* `handleEvent()` *Method*

```


<script type="text/javascript" language="JavaScript">
<!--
document.circle.handleEvent(Event.KEYDOWN);
// -->
</script>
```

Image.height

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
image.height
```

Description

The `height` property of the `Image` object specifies the height of the image in pixels. This property can only be set by the `height` attribute of the `<img>` tag.

Example

Listing 8.227 displays the value of the `height` property.

Listing 8.227 *Example of the* `height` *Property*

```
<html>


```

Listing 8.227 Continued

```
<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display the height of the image
document.write("The height of the image is ");
document.write(document.circle.height, " pixels.");

// -->
</script>

</html>
```

Image.hspace

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.hspace

Description

The hspace property of the Image object specifies the number of extra pixels that should appear on the left and right of the image. This property can only be set by the hspace attribute of the tag.

Example

Listing 8.228 demonstrates the hspace property. When the code sample is loaded, the JavaScript code will write to the screen the value of the Image hspace property.

Listing 8.228 Example of the hspace Property

```
<html>

Text to left of image.

Text to right of image.<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display value of hspace property
document.write("The hspace property of the image is ");
document.write(document.circle.hspace);
```

```
// -->
</script>

</html>
```

Image.lowsrc

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`image.lowsrc`

Description

The `lowsrc` property of the `Image` object specifies the URL of an alternate image to use on low-resolution displays. This property can only be set by the `lowsrc` attribute of the `<img>` tag.

Example

Listing 8.229 displays the URL of the low-resolution image. When the code sample is loaded into a browser, the JavaScript code will output the value of the `lowsrc` property.

Listing 8.229 *Example of the `lowsrc` Property*

```
<html>


<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display the low resolution image
document.write("The URL of the low resolution image is ");
document.write("<i><b>",document.circle.lowsrc,"</b></i>");

// -->
</script>

</html>
```

Image.name

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.name

Description

The name property of the Image object specifies the name of the image. This property can only be set by the name attribute of the tag.

Example

Listing 8.230 displays the name of the image. When the code sample is loaded into a browser, the JavaScript code will output the value of the name property.

Listing 8.230 Example using the name Property

```
<html>


<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display name of image
document.write("The name of the image is <i>");
document.write(document.circle.name,"</i>");

// -->
</script>

</html>
```

Image.onAbort

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

onAbort="command"

Description

The onAbort event defines a handler when the loading of the image is aborted.

Example

Listing 8.231 creates an alert box when the loading of the image is aborted.

Listing 8.231 Example of the `onAbort` Event

```

```

Image.onError

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+

Syntax

`onError="command"`

Description

The `onError` event handler is triggered when an error occurs while loading the image.

Example

Listing 8.232 creates an alert box if an error occurs while loading the image.

Listing 8.232 Example of the `onError` Event Handler

```

```

Image.onKeyDown

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`onKeyDown="command"`

Description

The `onKeyDown` event handler is triggered when a key is pressed down while the image is selected.

Example

Listing 8.233 defines an event handler for the `onKeyDown` event within an image.

Listing 8.233 Example of the `onKeyDown` Event

```
<input type="image" name="circle" src="circle.gif"
  ➤ onKeyDown="alert('An ONKEYDOWN event occurred!')">
```

Image.onKeyPress

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`onKeyPress="command"`

Description

The `onKeyPress` event defines a handler when a key is pressed down while the image is selected.

Example

Listing 8.234 defines an event handler for the `onKeyPress` event within an image.

Listing 8.234 *Example of the `onKeyPress` Event Handler*

```
<input type="image" name="circle" src="circle.gif"
  ➤ onKeyPress="alert('An ONKEYPRESS event occurred!')">
```

Image.onKeyUp

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`onKeyUp="command"`

Description

The `onKeyUp` event defines a handler when a key is pressed and then released while the image is selected.

Example

Listing 8.235 defines an event handler for the `onKeyUp` event within an image.

Listing 8.235 *Example of the `onKeyUp` Event*

```
<input type="image" name="circle" src="circle.gif"
  ➤ onKeyUp="alert('An ONKEYUP event occurred!')">
```

Image.onLoad

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`onLoad="command"`

Description

The onLoad event handler defines what should happen once the image has finished loading.

Example

Listing 8.236 displays a message once the image has finished loading.

Listing 8.236 *Example of the* onLoad *Event Handler*

```

```

Image.src

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.src

Description

The src property of the Image object specifies the URL of the image. This property can only be set by the src attribute of the tag.

Example

Listing 8.237 displays the URL of the image.

Listing 8.237 *Example of the* src *Property*

```
<html>

<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display the URL of the image
document.write("The URL of the image is ");
document.write("<i><b>",document.circle.src,"</b></i>");

// -->
</script>

</html>
```

Image.unwatch()

JavaScript 1.2+

Nav4+, NES3+

Syntax

```
image.unwatch(prop)
```

Description

The `unwatch()` method of the `Image` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to unwatch.

Example

Listing 8.238 shows how the `unwatch()` method is used to remove the watchpoint set on `myImg`.

Listing 8.238 Using the `unwatch()` Method on the `Image` Object

```
<html>

<br><br>

<script type="text/javascript" language="JavaScript">
<!--

// Function which is run when the value of myImg is changed.
// It alerts the user of the changes.
function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + 
    "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

myImg = document.myhome.name;
watch("myImg",alertme);
myImg = "Yourhome";
unwatch("myImg");
myImg = "theirhome";

// -->
</script>
</html>
```

Image.vspace

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

image.vspace

Description

The *vspace* property of the *Image* object specifies the number of extra pixels that should appear on the top and bottom of the image. This property can only be set by the *vspace* attribute of the `<img>` tag.

Example

Listing 8.239 demonstrates the *vspace* property. When the code sample is loaded into a browser, the JavaScript code will output the value of the *vspace* property.

Listing 8.239 *Example of the* *vspace* *Property*

```
<html>

Text at top of image.<br>
<br>
Text at bottom of image.<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display value of vspace property
document.write("The vspace property of the image is ");
document.write(document.circle.vspace);

// -->
</script>

</html>
```

Image.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

image.watch(prop, handler)

Description

The *watch()* method of the *Image* object is used to watch for changes to *Image* properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.240 shows how the `watch()` method is used to determine when the `name` property has been modified.

Listing 8.240 *Using the `watch()` Method of the Image Object*

```
<html>

<br><br>

<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" +
    oldValue + "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

myImg = document.myhome.name;
watch("myImg",alertme);
myImg = "Yourhome";
myImg = "theirhome";

// -->
</script>
</html>
```

Image.width

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

`image.width`

Description

The `width` property of the `Image` object specifies the width of the image in pixels. This property can only be set by the `width` attribute of the `<img>` tag.

Example

Listing 8.241 displays the value of the `width` property.

Listing 8.241 Example of the width Property

```
<html>



<br>

<script type="text/javascript" language="JavaScript">
<!--

//Display width of image
document.write("The width of the image is ");
document.write(document.circle.width, " pixels.");

// -->
</script>

</html>
```

Layer

JavaScript 1.2+
Nav4+

Syntax

Core client-side JavaScript object.

Description

The Layer object represents an object that contains a single document. A document can contain multiple layers, and thereby contain multiple documents. Layers are useful for displaying graphics which overlay each other within a browser. The Layer object has specific properties and methods associated with it, as shown in Table 8.19.

Table 8.19 Properties and Methods of the Layer Object

Type	Item	Description
Property	above	Specifies the layer above
	background	Refers to the background image of the layer
	below	Specifies the layer below
	bgColor	Refers to the background color of the layer
	clip.bottom	Refers to the bottom of the layer's clipping area
	clip.height	Refers to the height of the layer's clipping area
	clip.left	Refers to the left of the layer's clipping area
	clip.right	Refers to the right of the layer's clipping area
	clip.top	Refers to the top of the layer's clipping area
	clip.width	Refers to the width of the layer's clipping area

Table 8.19 Continued

Type	Item	Description
	document	The document object that contains the layer
	left	The x coordinate of the layer
	name	Refers to the name of the layer
	onBlur	Event handler when focus is removed from the layer
	onFocus	Event handler when focus is set to a layer
	onLoad	Event handler when a document is loaded in a layer
	onMouseOut	Event handler when the mouse cursor is removed from a layer's area
	onMouseOver	Event handler when the mouse cursor is moved over a layer's area
	pageX	The x coordinate relative to the document
	pageY	The y coordinate relative to the document
	parentLayer	The containing layer
	siblingAbove	The layer above in the zIndex
	siblingBelow	The layer below in the zIndex
	src	The source URL for the layer
	top	The y coordinate of the layer
	visibility	Specifies the visibility state of the layer
	zIndex	The relative z-order of this layer with respect to its siblings
	Method	
	captureEvents()	Specifies the event types to capture
	handleEvent()	Invokes handler for specified event
	load()	Loads a new URL
	moveAbove()	Moves the layer above another layer
	moveBelow()	Moves the layer below another layer
	moveBy()	Moves the layer to a specified position
	moveTo()	Moves the top-left corner of the window to the specified screen coordinates
	moveToAbsolute()	Changes the layer position to the specified pixel coordinates within the page
	releaseEvents()	Sets the layer to release captured events of the specified type
	resizeBy()	Resizes the layer by the specified height and width values
	resizeTo()	Resizes the layer to have the specified height and width values
	routeEvent()	Passes a captured event along the normal event hierarchy
	unwatch()	Removes a watchpoint on a Layer property
	watch()	Sets a watchpoint on a Layer property

Example

Listing 8.242 creates a layer. When the mouse cursor is moved over the Layer object, an alert message is displayed informing which layer the cursor was moved over.

Listing 8.242 *Example of the Layer Object*

```
<html>

<body>
<layer id="layer1" width="150" height="200" bgcolor="yellow" top="170"
left="200" visibility="show" onMouseOut='alert("This is layer 1")'>
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show" onMouseOut='alert("This is layer 2")'>
<center>Layer 2</center>
</layer>

</body>
</html>
```

Layer.above

JavaScript 1.2+
Nav4+

Syntax

layer.above

Description

The above property of the Layer object refers to the layer immediately above the layer in the zIndex. If this does not exist, the value is null.

Example

Listing 8.243 shows how the above property is used. Two layers are created and when the Get Above button is clicked, an alert box is displayed with the value of the above attribute.

Listing 8.243 *Example of the above Property*

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>
```

Listing 8.243 Continued

```
<layer ID="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Above" onClick='alert(document.layers.above)'\>
</form>

</body>
</html>
```

Layer.background

JavaScript 1.2+
Nav4+

Syntax

layer.background

Description

The background property of the Layer object refers to the background attribute of the <layer> tag. The background images for the layer object can be changed by setting the background.src property.

Example

Listing 8.244 shows an example of how to change the layer background image. Two layers are created and when the Change Background button is clicked, the change() function is called. The change() function sets the background property of layer1 to water.gif.

Listing 8.244 Example of the background Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function change(){
    document.layer1.background.src = "water.gif";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
```

```

<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Change Background" onClick='change()'>
</form>

</body>
</html>

```

Layer.below

JavaScript 1.2+
Nav4+

Syntax

layer.below

Description

The below property of the Layer object specifies the layer object immediately below the layer in the zIndex. If this does not exist, the below value is null.

Example

Listing 8.245 shows an example of the below property. Two layers are created using the <layer> tag. When the Get Below button is clicked, an alert box is displayed showing the value of the below property.

Listing 8.245 *Example of the below Property*

```

<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">

```

Listing 8.245 Continued

```
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Below" onClick='alert(document.layers.below)''>
</form>

</body>
</html>
```

Layer.bgColor

JavaScript 1.2+
Nav4+

Syntax

layer.bgColor

Description

The bgColor property of the Layer object represents the bgcolor attribute of the <layer> tag. The background color for the Layer object can be changed by setting the bgColor.src property. The color can only be changed if the background property is not set.

Example

Listing 8.246 shows how the bgColor property is used to change the background colors of the layer's objects.

Listing 8.246 Example of the bgColor Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function changes the color of layer1
function change1(){
    document.layer1.bgColor = "#FF00FA";
}

// function changes the color of Layer2
function change2(){
    document.layer2.bgColor="orange";
}
```



```
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Change Layer 1 Background" onClick='change1() '>
<br><br>
<input type="button" value="Change Layer 2 Background" onClick='change2() '>
</form>

</body>
</html>
```

Layer.captureEvents()

JavaScript 1.2+
Nav4+

Syntax

layer.captureEvents(event.type)

Description

The `captureEvents()` method of the `Layer` object is used to handle all events of a specific type.

Example

Listing 8.247 shows how the `captureEvents()` method is used. The `captureEvents()` method is used for `layer2` to listen for the `RESIZE` event. When this occurs, the `resize2()` function is called, which displays a message to the user.

Listing 8.247 Example of the `captureEvents()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
```

Listing 8.247 Continued

```
document.layer2.captureEvents(Event.RESIZE);

// function resizes layer2 and upon the resize event alerts the
// user that the layer was resized.
function resize2(){
    document.layer2.resizeTo(300,100);
    onResize=alert("Layer2 has been resized.");
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form>
Click the button to resize Layer 2
<br>
<input type="button" value="Resize Layer2" onClick='resize2()'>
</form>
</body>
</html>
```

Layer.clip.bottom

JavaScript 1.2+
Nav4+

Syntax

layer.clip.bottom

Description

The `clip.bottom` property of the `Layer` object refers to the bottom of the layer's clipping area.

Example

Listing 8.248 shows an example of how `clip.bottom` is used. Two layers are created using the `<layer>` tag. When the button is clicked, the `clip1` function is called, which clips the bottom of `layer1` by 100 pixels.

Listing 8.248 Example of the clip.bottom Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function clips the bottom of layer 1 by 100 pixels
function clip1(){
    document.layer1.clip.bottom = 100;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip bottom of Layer 1" onClick='clip1()''>
</form>

</body>
</html>

```

Layer.clip.height

JavaScript 1.2+
Nav4+

Syntax

`layer.clip.height`

Description

The `clip.height` property of the `Layer` object refers to the height of the layer's clipping area.

Example

Listing 8.249 shows how the `clip.height` property is used. Two layers are created using the `<layer>` tag. When the button is clicked, the `clip1()` function is called, which clips the height of `layer1` by 75 pixels.

Listing 8.249 Example of the clip.height Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function clips the height of layer 1 by 75 pixels
function clip1(){
    document.layer1.clip.height = 75;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip Height of Layer 1" onClick='clip1()''>
</form>

</body>
</html>
```

Layer.clip.left

JavaScript 1.2+
Nav4+

Syntax

layer.clip.left

Description

The `clip.left` property of the `Layer` object represents the left of the layer's clipping area.

Example

Listing 8.250 shows how the `clip.left` property is used. Two layers are created using the `<layer>` tag. When the button is clicked, the `clip1()` function is called, which clips the left side of `layer1` by 90 pixels.

Listing 8.250 Example of the clip.left Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function clips the left side of layer1 by 90 pixels
function clip1(){
    document.layer1.clip.left = 90;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip Left of Layer 1" onClick='clip1() '>
</form>

</body>
</html>

```

Layer.clip.right

JavaScript 1.2+
Nav4+

Syntax

layer.clip.right

Description

The `clip.right` property of the `Layer` object represents the right of the layer's clipping area.

Example

Listing 8.251 shows how the `clip.right` property is used to shorten the right side of the layer. Two layers are created using the `<layer>` tag. When the button is clicked, the `clip1()` function is called, which clips the right side of `layer1` by 110 pixels.

Listing 8.251 Example of the `clip.right` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function clips the right side of layer1 by 110 pixels
function clip1(){
    document.layer1.clip.right = 110;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip Right of Layer 1" onClick='clip1()''>
</form>

</body>
</html>
```

Layer.clip.top

JavaScript 1.2+
Nav4+

Syntax

`layer.clip.top`

Description

The `clip.top` property of the `Layer` object represents the top part of the layer's clipping area.

Example

Listing 8.252 uses the `clip.top` property to shorten the height of the layer. Two layers are created using the `<layer>` tag. When the button is clicked, the `clip1()` function is called, which clips the top of `layer1` by 88 pixels.

Listing 8.252 Example of the clip.top Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function clips the top of layer 1 by 88 pixels.
function clip1(){
    document.layer1.clip.top = 88;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip Top of Layer 1" onClick='clip1()'>
</form>

</body>
</html>

```

Layer.clip.width

JavaScript 1.2+
Nav4+

Syntax

layer.clip.width

Description

The clip.width property of the Layer object represents the width of the layer's clipping area.

Example

Listing 8.253 shows how the clip.width property is used to shorten the width of the layer. Two layers are created using the <layer> tag. When the button is clicked, the clip1() function is called, which clips the width of layer1 by 60 pixels.

Listing 8.253 Example of the clip.width Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function changes the width of Layer 1
function clip1(){
    document.layer1.clip.width = 60;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Clip Width of Layer 1" onClick='clip1() '>
</form>

</body>
</html>
```

Layer.document

JavaScript 1.2+
Nav4+

Syntax

`layer.document`

Description

The document property of the Layer object references the Document object contained in the layer.

Example

Listing 8.254 shows how the document property is used. When the button is clicked, the `getInfo()` function is called. This displays the name of the document in `layer1`.

Listing 8.254 Example of the document Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function displays an alert box indicating the name of Layer 1 document
function getInfo(){
    alert("The name of Layer 1's document is: " +
    ➤ document.layer1.document.name);
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer1 document info" onClick='getInfo()'>
</form>

</body>
</html>

```

Layer.handleEvent()

JavaScript 1.2+
Nav4+

Syntax

layer.handleEvent(*event*)

Description

The `handleEvent` method of the `Layer` object determines what type of event occurred and passes the event to the object's appropriate event handler.

Example

Listing 8.255 shows how to use the `handleEvent` method. The document is set up to capture any FOCUS events. When one occurs, it is sent to the `handle()` function, which

passes it to the default handler for layer2 object. This calls the `displayMsg()` function for any FOCUS events that occur.

Listing 8.255 Example of the `handleEvent()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// sets up the document to capture FOCUS events
document.captureEvents(Event.FOCUS);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handle(evnt){
    window.document.layer2.handleEvent(evnt);
}

function displayMsg(){
    alert("Focus event occurred.");
}

// This registers the handle function as the event handler for the
// FOCUS event.
document.onFocus = handle;
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top=100
left="70" visibility="show" onFocus='displayMsg() '>
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer1 document info" onClick='getInfo() '>
</form>

</body>
</html>
```

Layer.left

JavaScript 1.2+
Nav4+

Syntax

`layer.left`

Description

The `left` property of the `Layer` object represents the x-coordinate position of the layer within the document. Changing this property can move the layer left or right.

Example

Listing 8.256 shows how the `left` property is used to move the layer from left to right across the screen. When the button is clicked, the `move()` function is called, which moves `layer1` to the left by 60 pixels.

Listing 8.256 Example of the `left` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function moves Layer 1 to the left
function move(){
    document.layer1.left = document.layer1.left + 60;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 1" onClick='move()''>
</form>

</body>
</html>
```

Layer.load()

JavaScript 1.2+
Nav4+

Syntax

layer.load(src, width)

Description

The `load()` method of the `Layer` object is used to load a new document in the layer. As described in the syntactical definition, this method takes the URL of the source of the layer and the width.

Example

Listing 8.257 shows how to use the `load()` method. When the button is clicked, the `onClick` event handler loads the `tmp.html` file.

Listing 8.257 *Example of the `load()` Method*

```
<html>
<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Load Layer 2"
➡ onClick='document.layer2.load("tmp.html", 400)'\>
</form>

</body>
</html>
```

Layer.moveAbove()

JavaScript 1.2+
Nav4+

Syntax

layer.moveAbove(layername)

Description

The `moveAbove()` method of the `Layer` object is used to move the current layer above another specified layer. The parameter, *layername*, is the `Layer` object that gets moved back.

Example

Listing 8.258 an example of how the `moveAbove()` method is used. When the button is clicked, the `onClick` event handler handles the `CLICK` event and moves `Layer1` above `Layer2`.

Listing 8.258 Example of the `moveAbove()` Method

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="150" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 1 above layer 2"
onClick='document.layer1.moveAbove(document.layer2) '>
</form>

</body>
</html>
```

Layer.moveBelow()

JavaScript 1.2+
Nav4+

Syntax

`layer.moveBelow(layername)`

Description

The `moveBelow()` method of the `Layer` object is used to move a `Layer` object below another specified layer. The function takes a single parameter, *layername*, representing the `Layer` object that gets moved forward.

Example

Listing 8.259 shows how the `moveBelow()` method is used to move Layer2 below Layer1.

Listing 8.259 *Example of the `moveBelow()` Method*

```
<html>
<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="150" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 2 below layer 1"
onClick='document.layer2.moveBelow(document.layer1)''>
</form>

</body>
</html>
```

Layer.moveBy()

JavaScript 1.2+
Nav4+

Syntax

`layer.moveBy(x,y)`

Description

The `moveBy()` method of the Layer object moves the layer object to the right and down from its current position. The method takes two parameters, *x* and *y*. The *x* parameter refers to the number of pixels the layer is moved to the right. The *y* parameter refers to the number of pixels the layer is moved down.

Example

Listing 8.260 shows how the `moveBy()` method is used. When the button is clicked, the `onClick` event handler uses the `moveBy` method to move `layer2` 50 pixels to the right and 30 pixels down.

Listing 8.260 Example of the `moveBy()` Method

```

<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="150" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 2"
  ⚡ onClick='document.layer2.moveBy(50,30) ' >
</form>

</body>
</html>

```

Layer.moveTo()

JavaScript1.2+
 Nav4+

Syntax

`Layer.moveTo(x,y)`

Description

The `moveTo()` method of the `Layer` object moves the top-left corner of the `Layer` object to the specified screen coordinates. The `x` parameter refers to an integer representing the top edge of the window in screen coordinates. The `y` parameter refers to an integer representing the left edge of the window in screen coordinates.

Example

Listing 8.261 shows how the `moveTo()` method is used. Two layers are created using the `<layer>` tag. When the button is clicked, the `onClick` event handler for `layer2` invokes the `moveTo` method, which moves the layer to the screen coordinates 100, 200.

Listing 8.261 Example of the `moveTo()` Method

```

<html>

<body>

<layer id="layer1" width="200" height="200" BGCOLOR="yellow" top="170"
left="150" visibility="show">

```

Listing 8.261 Continued

```
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" BGCOLOR="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 2"
➡ onClick='document.layer2.moveTo(100,200) '>
</form>

</body>
</html>
```

Layer.moveToAbsolute()

JavaScript 1.2+
Nav4+

Syntax

layer.moveToAbsolute(*x*,*y*)

Description

The `moveToAbsolute()` method of the `Layer` object moves the upper-left corner of the layer to the specified position. This position is relative to the top-level document. The method takes two parameters, *x* and *y*. The *x* parameter refers to the number of pixels the layer is moved to the right. The *y* parameter refers to the number of pixels the layer is moved down.

Example

Listing 8.262 an example of how the `moveToAbsolute()` method is used. When the button is clicked, the upper-left corner of the second layer is moved to position 350x 400y.

Listing 8.262 Example of the `moveToAbsolute()` Method

```
<html>
<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="150" visibility="show">
<center>Layer 1</center>
</layer>
```



```

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 2"
onClick='document.layer2.moveToAbsolute(350,400) '>
</form>

</body>
</html>

```

Layer.name

JavaScript 1.2+
Nav4+

Syntax

layer.name

Description

The `name` property of the `Layer` object refers to the `name` or `ID` attribute of the `<layer>` tag.

Example

Listing 8.263 shows how to access the layer's name.

Listing 8.263 Example of the `name` Property

```

<html>
<body>

<layer id="Tarzan" width="200" height="200" bgcolor="yellow" top="170"
left="250" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="Jane" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer 1 Name"
onClick='alert("The name of this layer is: " + document.layer1.name) '>
<br><br><br>

```

Listing 8.263 Continued

```
<input type="button" value="Get Layer 2 Name"
onClick='alert("The name of this layer is: " + document.layer2.name) ' >
</form>

</body>
</html> >
```

Layer.onBlur

JavaScript 1.2+
Nav4+

Syntax

```
onBlur="command"
```

Description

The onBlur event handler of the Layer object handles the event when the focus is removed from the Layer object.

Example

Listing 8.264 shows how the onBlur event handler is used. When focus is removed from layer1, the onBlur event handler calls the showMsg() function.

Listing 8.264 Example of the onBlur Event Handler

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
function showMsg(){
    document.form1.text1.value = "Focus was removed from Layer 1";
}
// -->
</script>

<layer id="layer1" width=200 height=200 bgcolor="yellow" top="170"
left="200" visibility="show"onBlur='showMsg()' >
<center>Layer 1</center>
</layer>

<layer id="layer2" width=150 height=160 bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>
```

```

<form name="form1">
Click on Layer 1 and the click on Layer 2.
<br><br>
<input type="text" name="text1" size="40">
</form>

</body>
</html>

```

Layer.onFocus

JavaScript 1.2+
Nav4+

Syntax

```
onFocus="command"
```

Description

The onFocus event handler for the Layer object notifies you when the focus is set on a layer object.

Example

Listing 8.265 shows how the onFocus event handler is used to detect when focus is set on Layer 1.

Listing 8.265 *Example of the* onFocus *Event Handler*

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--
function showMsg(){
    document.form1.text1.value = "Focus set on Layer 1";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show"onFocus='showMsg()'>
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

```

Listing 8.265 Continued

```
<form name="form1">
Click on Layer 2 and then click on layer 1.
<br><br>
<input type="text" name="text1" size="40">
</form>

</body>
</html>
```

Layer.onLoad

JavaScript 1.2+
Nav4+

Syntax

onLoad="command"

Description

The onLoad property of the Layer object is an event handler that notifies you when the layer's contents are being loaded.

Example

Listing 8.266 shows how the onLoad event handler is used.

Listing 8.266 Example of the onLoad Event Handler

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function showMsg(){
    document.form1.text1.value = "Layer contents being loaded";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show"onLoad='showMsg() '>
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70"visibility="show">
<center>Layer 2</center>
</layer>
```

```

<form name="form1">
<input type="text" name="text1" size="40">
<br><br>
<input type="button" value="Load Layer 1"
  ➤ onClick='document.layer1.load("tmp.html",300) ' >
</form>

</body>
</html>

```

Layer.onMouseOut

JavaScript 1.2+

Nav4+

Syntax

onMouseOut="command"

Description

The onMouseOut property of the Layer object is an event handler that notifies you when the mouse cursor has moved out of the layer region.

Example

Listing 8.267 shows how the onMouseOut event handler is used. When the the mouse cursor is moved over Layer 1 and then is moved out of the Layer 1 region, the event handler invokes the showMsg1() function to alert the user.

Listing 8.267 Example of the onMouseOut Event Handler

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function showMsg1(){
    document.form1.text1.value = "Mouse Moved out of Layer 1 area.";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show"onMouseOut=' showMsg1()' >
<center>Layer 1</center>
</layer>

```

Listing 8.267 Continued

```
<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="text" name="text1" size="40">
<br><br>
</form>

</body>
</html>
```

Layer.onMouseOver

JavaScript 1.2+
Nav4+

Syntax

`onMouseOver="command"`

Description

The `onMouseOver` event handler of the `Layer` object handles the events when the mouse cursor is moved over the layer area.

Example

Listing 8.268 shows an example of the `onMouseOver` event handler. When the mouse cursor is moved over Layer 2, the event handler invokes the `showMsg1()` function to alert the user.

Listing 8.268 Example of the `onMouseOver` Event Handler

```
<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// function displays a message everytime the mouse is moved over Layer 2
function showMsg1(){
    document.form1.text1.value = "Mouse Moved over Layer 2 area.";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
```

```

<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show" onMouseOver='showMsg1()' '>
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="text" name="text1" size="40">
<br><br>
</form>

</body>
</html>

```

Layer.pageX

JavaScript 1.2+
Nav4+

Syntax

layer.pageX

Description

The pageX property of the Layer object represents the x-coordinate position of the layer relative to the top-level document.

Example

Listing 8.269 shows how to use the pageX property. When the button is clicked, the value of the pageX property is displayed.

Listing 8.269 *Example of the pageX Property*

```

<html>
<body>

<script type="text/javascript" language="JavaScript">
<!--

// function increases the size of Layer 1 pageX property
function size(){
    document.layer1.pageX = document.layer1.pageX +20;
}
// -->
</script>

```

Listing 8.269 Continued

```
<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Change Layer 1 X-coordinate" onClick=size()>
</form>

</body>
</html>
```

Layer.pageY

JavaScript 1.2+
Nav4+

Syntax

layer.pageY

Description

The `pageY` property of the `Layer` object represents the y-coordinate position of the layer relative to the top-level document.

Example

Listing 8.270 shows how to manipulate the `pageY` property. When the button is clicked, the `size` function is called, which adds 20 pixels to the `pageY` attribute of `layer1`.

Listing 8.270 Example of the `pageY` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function changes the value of the PageY attribute
function size(){
    document.layer1.pageY = document.layer1.pageY +20;
}
```



```
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width=150 height=160 bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Change Layer 1 Y-coordinate" onClick='size() '>
</form>

</body>
</html>
```

Layer.parentLayer

JavaScript 1.2+

Nav4+

Syntax

layer.parentLayer

Description

The parentLayer property of the Layer object represents the window or Layer object that contains the current Layer object.

Example

Listing 8.271 shows how to use the parentLayer. The function getname() gets and returns the name of the parent layer.

Listing 8.271 *Example of the parentLayer Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
```

Listing 8.271 Continued

```
// function gets the name of the parent layer
function getname(){
    document.form1.text1.value = document.layer1.parentLayer.name;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width=150 height=160 bgcolor="green" top="100"
left="70" visibility="show">'>
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer 1 parent name" onClick='getname()'>
<br><br>
<input type="text" name="text1" size="40">
</form>

</body>
</html>
```

Layer.releaseEvents()

JavaScript 1.2+
Nav4+

Syntax

layer.releaseEvents(*event*)

Description

The `releaseEvents()` method of the `Layer` object specifies that the `Layer` object should no longer capture events of the specified type.

Example

Listing 8.272 is an example of how the `releaseEvents()` method is used. When the `FOCUS` event is captured, it is handled by the `handle()` method. After this is done, executing the event is released using the `releaseEvents()` method.

Listing 8.272 Example of the `releaseEvents()` Method

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets up the document to capture FOCUS events
document.captureEvents(Event.FOCUS);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The layer got a FOCUS event and calls releaseEvents");
    return true;
}

function releaseFocus(){
    document.layer2.releaseEvents(Event.FOCUS);
}

// This registers the handle function as the event handler for the
// FOCUS event.
document.onFocus = handle;

// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer1" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

</body>
</html>

```

Layer.resizeBy()

JavaScript 1.2+
Nav4+

Syntax

`layer.resizeBy(x,y)`

Description

The `resizeBy()` method of the `Layer` object resizes the layer by a relative size. The `x` parameter refers to the number of pixels the layer width is increased. The `y` parameter refers to the number of pixels the layer height is increased.

Example

Listing 8.273 shows how to resize a `Layer` object. When the button is clicked, the `onClick` event handler executes the `resizeBy()` method to resize `layer2`.

Listing 8.273 *Example of the* `resizeBy()` *Method*

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Resize Layer 2"
  ⚡ onClick='document.layer2.resizeBy(30, 50)'\>
<br>
</form>

</body>
</html>
```

Layer.resizeTo()

JavaScript 1.2+
Nav4+

Syntax

`layer.resizeTo(x,y)`

Description

The `resizeTo()` method of the `Layer` object resizes the `Layer` object to the specified value. The `x` parameter refers to the number of pixels by which the layer width is increased. The `y` parameter refers to the number of pixels by which the layer height is increased.

Example

Listing 8.274 shows how to resize a layer. When the button is clicked, the `onClick` event handler uses the `resizeTo()` method to resize `layer2`.

Listing 8.274 *Example of the* `resizeTo()` *Method*

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Resize Layer 2"
  ➤ onClick='document.layer2.resizeTo(100, 300)'\>
<br>
</form>

</body>
</html>
```

Layer.routeEvent()

JavaScript 1.2+
Nav4+

Syntax

```
layer.routeEvent(event)
```

Description

The `routeEvent()` method of the `Layer` object reroutes a captured event to another event handler. The method takes a single parameter representing the event to be routed.

Example

Listing 8.275 shows an example of the `routeEvent()` method. When the focus event occurs, it is captured and the event is rerouted.

Listing 8.275 Example of the routeEvent() Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// sets up the document to capture FOCUS events
document.captureEvents(Event.FOCUS);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    alert("The layer got a FOCUS event and executes routeEvent");
    document.routeEvent(evt);
    alert("Event has been routed");
    return true;
}

// This registers the handle function as the event handler for the
// FOCUS event.
document.onFocus = handle;

// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

</body>
</html>
```

Layer.siblingAbove

JavaScript 1.2+
Nav4+

Syntax

layer.siblingAbove

Description

The `siblingAbove` property of the `Layer` object refers to the sibling layer immediately above the current layer in the `zIndex`. If this doesn't exist, `null` is returned.

NOTE

Any child of a parent layer is considered a sibling.

Example

Listing 8.276 shows how to get the `siblingAbove` property value. When the button is clicked, the `siblingAbove` value is displayed.

Listing 8.276 Example of the `siblingAbove` Property

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer1" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Sibling Above Layer 1"
onClick='alert(document.layer1.siblingAbove)''>
<br>
</form>

</body>
</html>
```

Layer.siblingBelow

JavaScript 1.2+
Nav4+

Syntax

`layer.siblingBelow`

Description

The `siblingBelow` property of the `Layer` object refers to the sibling layer immediately below the current layer in the `zIndex`. If this doesn't exist, `null` is returned.

Example

Listing 8.277 shows the `siblingBelow` property. When the button is clicked, the `siblingBelow` value is displayed.

Listing 8.277 *Example of the `siblingBelow` Property*

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer1" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Sibling Above Layer 1"
onClick='alert(document.layer1.siblingBelow) '>
<br>
</form>

</body>
</html>
```

Layer.src

JavaScript 1.2+
Nav4+

Syntax

layer.src

Description

The `src` property of the `Layer` object represents the source URL of a particular layer.

Example

Listing 8.278 shows how the `src` property is used. When the button is clicked, the `src` property is displayed.

Listing 8.278 Example of the `src` Property

```

<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer 1 src"
  ⚡onClick='alert(document.layer1.src)'\>
</form>

</body>
</html>

```

Layer.top

JavaScript 1.2+
Nav4+

Syntax

`layer.top`

Description

The `top` property of the `Layer` object represents the y-coordinate position of the layer relative to the top-level document. Increasing or decreasing this value can move the `Layer` object up or down.

Example

Listing 8.279 uses the `top` property to move the layer up. When the button is clicked, the `move()` function is executed, which will move the layer up by 10 pixels.

Listing 8.279 Example of the `top` Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

```

Listing 8.279 Continued

```
function move(){
    document.layer1.top = document.layer1.top - 10;
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Move layer 1 Up" onClick='move() '>
</form>

</body>
</html>
```

Layer.unwatch()

JavaScript 1.2+
Nav4+

Syntax

layer.unwatch(*prop*)

Description

The `unwatch()` method of the `Layer` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to `unwatch`.

Example

Listing 8.280 shows how the `unwatch()` method is used to disable the watchpoint set on the `name` property of the `Layer` object.

Listing 8.280 Using the `unwatch()` Method of the `Layer` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
```

```

    document.writeln(id + " changed from " + "<b>" + oldValue +
    "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.layers.name;
    watch("myProp",alertme);
    myProp = "red_layer";
    unwatch("myProp");
    myProp = "blue_layer";
}

// -->
</script>
<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer ID="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Enable/Disable watchpoint" onClick='setWatch() '>
</form>

</body>
</html>

```

Layer.visibility

JavaScript 1.2+
Nav4+

Syntax

layer.visibility

Description

The *visibility* property of the *Layer* object controls whether the layer is displayed or hidden. Valid values for this property are: *hide*, *show*, and *inherit*. This property also represents the HTML *visibility* attribute of the *<layer>* tag.

Example

Listing 8.281 shows how the `visibility` property is used to hide a `Layer` object.

Listing 8.281 *Example of the `visibility` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function hide(){
    document.layer2.visibility =" hide";
}
// -->
</script>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Hide Layer 2" onClick='hide()''>
</form>

</body>
</html>
```

Layer.watch()

JavaScript 1.2+
Nav4+

Syntax

`layer.watch(prop, handler)`

Description

The `watch()` method of the `Layer` object is used to watch for changes to `Layer` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.282 shows how the `watch()` method is used to set a watchpoint on the `name` property. When the name is changed, the `alertme` function is called to display the changes to the browser.

Listing 8.282 Using the `watch()` Method of the `Layer` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    ↪ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.layers.name;
    watch("myProp",alertme);
    myProp = "red_layer";
    myProp = "blue_layer";
}

// -->
</script>
<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer ID="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Enable watchpoint" onClick='setWatch() '>
</form>

</body>
</html>
```

Layer.zIndex

JavaScript 1.2+
Nav4+

Syntax

layer.zIndex

Description

The *zIndex* property of the *Layer* object represents the stacking order of the layers.

Example

Listing 8.283 shows an example of the *zIndex* property. When the button is clicked, an alert box is displayed showing the value of *zIndex*.

Listing 8.283 *Example of the *zIndex* Property*

```
<html>

<body>

<layer id="layer1" width="200" height="200" bgcolor="yellow" top="170"
left="200" visibility="show">
<center>Layer 1</center>
</layer>

<layer id="layer2" width="150" height="160" bgcolor="green" top="100"
left="70" visibility="show">
<center>Layer 2</center>
</layer>

<form name="form1">
<input type="button" value="Get Layer 2 zIndex"
  ➤ onClick='alert(document.layer2.zIndex) ' >
</form>

</body>
</html>
```

Link

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The `Link` object represents an HTML hypertext link. This can be an image, text, or pre-defined area within the Web page. All HTML links are stored in a `links[]` array. The `Link` object has specific properties and methods associated with it, as shown in Table 8.20.

Table 8.20 *Properties and Methods of the `Link` Object*

Type	Item	Description
Property	<code>handleEvent</code>	Event Handler
	<code>hash</code>	Represents an anchor name in the URL for the link, which begins with the <code>#</code> character
	<code>host</code>	Represents the host portion of the URL associated with a link
	<code>hostname</code>	Represents the hostname portion of the URL associated with a link
	<code>href</code>	Represents the complete URL associated with a link
	<code>onClick</code>	Event handler for mouse click events
	<code>onDbClick</code>	Event handler for double mouse click events
	<code>onKeyDown</code>	Event handler for pressing a key down on a <code>Link</code> object
	<code>onKeyPress</code>	Event handler for pressing a key on the <code>Link</code> object
	<code>onKeyUp</code>	Event handler for releasing a key on the <code>Link</code> object
	<code>onMouseDown</code>	Event handler for pressing the mouse button down on the link
	<code>onMouseOut</code>	Event handler for moving the mouse cursor away from the link
	<code>onMouseOver</code>	Event handler for moving the mouse cursor over the link
	<code>onMouseUp</code>	Event handler for releasing the mouse button on the link
	<code>pathname</code>	Represents the pathname portion of the link URL
	<code>port</code>	Represents the port portion of the link URL
	<code>protocol</code>	Specifies the protocol portion of the link URL
	<code>search</code>	Represents the query portion of the link URL
	<code>target</code>	Represents the name of the Window object in which the link is displayed
	<code>text</code>	The text used to create the link
Method	<code>unwatch()</code>	Removes a watchpoint on a <code>Link</code> property
	<code>watch()</code>	Sets a watchpoint on a <code>Link</code> property

Example

Listing 8.284 shows how to use the `Link` object. When the link is clicked, the text of the link is displayed.

Listing 8.284 Example of the Link Object

```
<html>

<body>

Click on the link to go to the site.
<br><br>
<a href="http://www.sampublishing.com"
onClick='alert("Go to:" + document.links[0].text)'+> Sams Publishing</a>

</body>
</html>
```

Link.handleEvent()

JavaScript 1.2+
Nav4+

Syntax

```
link.handleEvent(event)
```

Description

The `handleEvent()` method invokes the event handler for the `Link` object.

Example

Listing 8.285 shows how to access the `handleEvent()` method. All click events are handled by the first link in the document.

Listing 8.285 Example of the handleEvent() Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets up the document to capture CLICK events
document.captureEvents(Event.CLICK);

// function that handles the specific event. The evt parameter refers to
// the event object.
function handle(evt){
    document.form1.links[0].handleEvent(Event.CLICK);
}

function displayMsg(){
    alert("Click event occurred.");
}

}
```



```
// This registers the handle function as the event handler for the
// CLICK event.
document.onClick = handle;

// -->
</script>

<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onClick='displayMsg()' '>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br>
</form>

</body>
</html>
```

Link.hash

JavaScript 1.0+
Nav2+, Opera3+

Syntax

link.hash

Description

The hash property represents a portion of the currently displayed URL.

Example

Listing 8.286 shows an example of how the hash property is used. When the button is clicked, the `showHash()` function displays the value of the hash property for the link.

Listing 8.286 *Example of the hash Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function shows the hash value of the link
function showHash(){
    document.form1.text1.value = document.links[0].hash;
}
// -->
</script>
```

Listing 8.286 Continued

Click the button to see the hash for the URL

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Hash value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="hash" value="Get hash value" onClick='showHash()'>
<br><br>
</form>

</body>
</html>
```

Link.host

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

link.host

Description

The host property represents the host portion of the URL.

Example

Listing 8.287 shows how to use the host property. When the button is clicked, the `showHost()` function is called to show the value of the host property.

Listing 8.287 Example of the *host* Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function shows the host value
function showHost(){
    document.form1.text1.value = document.links[0].host;
}
// -->
</script>
```

Click the button to see the host for the URL

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
```

```

http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Host value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="host" value="Get Host Value" onClick='showHost()'>
<br>
</form>

</body>
</html>

```

Link.hostname

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Link.hostnamename

Description

The hostname property of the Link object represents the hostname portion of the link's URL.

Example

Listing 8.288 shows an example of how the hostname property is used. When the button is clicked, the showHostname() function is called, which displays the value of the hostname property for the link.

Listing 8.288 *Example of the* hostname *Property*

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the hostname for the URL
function showHostname(){
    document.form1.text1.value = document.links[0].hostname;
}
// -->
</script>

```

Click the button to see the hostname for the URL

```

<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>

```

Listing 8.288 Continued

```
<br><br>
Host value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="host" value="Get Hostname Value"
  ➤ onClick='showHostname()'>
<br>
</form>

</body>
</html>
```

Link.href

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

link.href

Description

The href property of the Link object represents the whole URL for the link.

Example

Listing 8.289 shows how to use the href property. When the button is clicked, the showhref() function displays the value of the href property of the link.

Listing 8.289 Example of the href Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the value of the href property in the text box.
function showhref(){
    document.form1.text1.value = document.links[0].href;
}
// -->
</script>
```

Click the button to see the href property for the URL

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
```

```
Host value: <input type="text" name="text1" size="45">
<br><br>
<input type="button" name="href" value="Get href Value" onClick='showhref() '>
<br>
</form>

</body>
</html>
```

Link.onClick

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`onClick="command"`

Description

The `onClick` event handler for the `Link` object is used to handle the event when the mouse cursor is clicked on the link.

Example

Listing 8.290 shows how to determine a mouse click using the `onClick` event handler.

Listing 8.290 *Example of the `onClick` Event Handler*

```
<html>

<body>

Click the link.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
➡ onClick='alert("You clicked the link")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>
```

Link.onDbClick

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

`onDbClick="command"`

Description

The `onDb1Click` event handler of the `Link` object is used to handle the event when the mouse cursor is double-clicked on the link.

Example

Listing 8.291 shows how to use the `onDb1Click` event handler. When the link is double-clicked, the event handler alerts the user indicating that the double-click action occurred.

Listing 8.291 *Example of the `onDb1Click` Event Handler*

```
<html>

<body>

Click the link.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onDb1Click='alert("You double-clicked the link")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>
```

Link.onKeyDown

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

```
onKeyDown="command"
```

Description

The `onKeyDown` event handler for the `Link` object is used to handle the event when a key is pressed down while the focus is on the link.

Example

Listing 8.292 shows how the `onKeyDown` event handler operates. When a key is pressed down, the event handler alerts the user about the action that occurred.

Listing 8.292 *Example of the `onKeyDown` Event Handler*

```
<html>

<body>
```

Click the link.

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onKeyDown='alert("You pressed the key DOWN on the link")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>
```

Link.onKeyPress

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

onKeyPress="command"

Description

The onKeyPress event handler for the Link object is used to handle the event if a key is pressed when the focus is set on the link.

Example

Listing 8.293 shows how the onKeyPress event handler is used. The onKeyPress event handler is invoked any time a key is pressed on the link.

Listing 8.293 *Example of the onKeyPress Event Handler*

```
<html>

<body>

Click the link.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onKeyPress='alert("You pressed the key on the link")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>
```

Link.onKeyUp

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

`onKeyUp="command"`

Description

The `onKeyUp` event handler for the `Link` object is used to handle events in which a key is pressed and then released.

Example

Listing 8.294 shows an example of how the `onKeyUp` event handler is used. The `onKeyUp` event handler is invoked any time a key is pressed and released on the link.

Listing 8.294 Example of the `onKeyUp` Event Handler

```
<html>

<body>

Highlight the link, press a key and let it up.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onKeyUp='alert("You let the key up")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>
```

Link.onMouseDown

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

`onMouseDown="command"`

Description

The `onMouseDown` event handler of the `Link` object is used to handle the event when the mouse button is pressed down while the mouse pointer is over the link.

Example

Listing 8.295 shows how the `onMouseDown` event handler is used. The `onMouseDown` event handler is invoked any time the mouse button is pressed down while the cursor is on the link.

Listing 8.295 Example of the `onMouseDown` Event Handler

```

<html>

<body>

Click the mouse button while the cursor is on the link.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onMouseDown='alert("The mouse button was pressed DOWN.")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
</form>

</body>
</html>

```

Link.onMouseOut

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
onMouseOut="command"
```

Description

The `onMouseOut` event handler of the `Link` object is used to handle the event when the mouse cursor is moved away from the link.

Example

Listing 8.296 shows how the `onMouseOut` event handler is used.

Listing 8.296 Example of the `onMouseOut` Event Handler

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// displays message when the MouseEvent occurs.
function showMsg(){
    document.form1.text1.value = "The Mouse cursor was removed from the link";
}
// -->
</script>

```

Listing 8.296 Continued

Click the mouse button while the cursor is on the link.

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onMouseOut='showMsg()'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
<input type="text" name="text1" size="50">
</form>

</body>
</html>
```

Link.onMouseOver

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onMouseOver="command"
```

Description

The `onMouseOver` event handler of the `Link` object handles the event when the mouse cursor is moved over the HTML link.

Example

Listing 8.297 shows how the `onMouseOver` event handler is used to display a message when the mouse cursor is moved over the link.

Listing 8.297 Example of the `onMouseOver` Event Handler

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets the text in the textbox to display the message.
function showMsg(){
    document.form1.text1.value = "The Mouse cursor was moved over the link";
}

// -->
</script>
```

Move the mouse cursor over the link.

```
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onMouseOver='showMsg()'>
```

```
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
<input type="text" name="text1" size="50">
</form>

</body>
</html>
```

Link.onMouseUp

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

`onMouseUp="command"`

Description

The `onMouseUp` event handler for the `Link` object is used to handle the event when the mouse button is pressed on the link and then released.

Example

Listing 8.298 shows how the `onMouseUp` event handler is used.

Listing 8.298 Example of the `onMouseUp` Event Handler

```
<html>

<body>

Click the mouse button while the cursor is on the link.
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar"
onMouseUp='alert("The Mouse button was let up")'>
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
<input type="text" name="text1" size=50>
</form>

</body>
</html>
```

Link.pathname

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`link.pathname`

Description

The `pathname` property of the `Link` object represents the `pathname` portion of the link URL.

Example

Listing 8.299 shows how the `pathname` property is used. When the button is clicked, the `showpathname` function is executed. This displays the `pathname` for the link.

Listing 8.299 *Example of the* `pathname` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the pathname for the specified URL
function showpathname(){
    document.form1.text1.value = document.links[0].pathname;
}
// -->
</script>
```

Click the button to see the `pathname` for the URL

```
<form name="form1">
<a href="http://www.sampublishing.com:80/tmp/foo.html?something#foobar">
http://www.a.com:80/tmp/foo.html?something#foobar</a>
<br><br>
Pathname value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="path" value="Get Pathname Value"
  ⚡ onClick='showpathname()'>
<br>
</form>

</body>
</html>
```

Link.port

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`link.port`

Description

The port property of the Link object represents the port number in the URL. This is not always present in all URLs.

Example

Listing 8.300 shows how to obtain the port number if available. When the button is clicked, the port number for the link is displayed.

Listing 8.300 Accessing the port Number

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the port number of the URL
function showport(){
    document.form1.text1.value = document.links[0].port;
}
// -->
</script>

Click the button to see the port number for the URL
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Port value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="port" value="Get Port Number" onClick='showport() '>
<br>
</form>

</body>
</html>
```

Link.protocol

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Link.protocol

Description

The protocol property of the Link object represents the protocol being used in the current Web browser. This is the first piece of text in the URL.

Example

Listing 8.301 shows how you can get the protocol type being used. When the button is clicked, the protocol for the link is displayed.

Listing 8.301 *Example of the* protocol *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the type of protocol being used for the URL
function showproto(){
    document.form1.text1.value = document.links[0].protocol;
}
// -->
</script>

Click the button to see the protocol used for the URL
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Protocol value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="proto" value="Get Protocol" onClick='showproto()'>
<br>
</form>

</body>
</html>
```

Link.search

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

link.search

Description

The search property of the Link object represents the query portion of the URL (if available). This includes the leading question mark (?).

Example

Listing 8.302 shows how the search property can be determined. When the button is clicked, the showsearch() function displays the search value of the link.

Listing 8.302 Example of the `search` Property

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// display the information being queried
function showsearch(){
    document.form1.text1.value = document.links[0].search;
}
// -->
</script>

Click the button to see the search portion of the URL
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Search value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="search" value="Get Search portion"
  ⚡ onClick='showsearch()'>
<br>
</form>

</body>
</html>

```

Link.target

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`link.target`

Description

The `target` property of the `Link` object represents the name of the window in which the URL is to be displayed.

Example

Listing 8.303 shows how the `target` property is used. When the button is clicked, the `showtarget()` function displays the `target` value of the link.

Listing 8.303 Example of the `target` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the name of the window in which the link will be displayed.
function showtarget(){
    document.form1.text1.value = document.links[0].target;
}
// -->
</script>

Click the button to see the target of the URL
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
http://www.sampublishing.com:80/foo?something#foobar</a>
<br><br>
Target value: <input type="text" name="text1" size="20">
<br><br>
<input type="button" name="tar" value="Get Target" onClick='showtarget()'>
<br>
</form>

</body>
</html>
```

Link.text

JavaScript 1.2+
Nav4+

Syntax

`link.text`

Description

The text property of the Link object is used to get the text value of the link.

Example

Listing 8.304 shows how to use the text property. When the button is clicked, the showtext function displays the text value of the link.

Listing 8.304 Example of the `text` Property

```
<html>

<body>
```



```

<script type="text/javascript" language="JavaScript">
<!--
// function to display the value of the text property
function showtext(){
    document.form1.text1.value = document.links[0].text;
}
// -->
</script>

```

Click the button to see the text value of the link.

```

<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
Link to Something</a>
<br><br>
Text value: <input type="text" name="text1" size="50">
<br><br>
<input type="button" name="txt" value="Get Text Value" onClick='showtext() '>
<br>
</form>

</body>
</html>

```

Link.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

Link.unwatch(prop)

Description

The `unwatch()` method of the `Link` object will remove a watchpoint on a property, *prop*, set by the `watch()` method.

Example

Listing 8.305 shows how the `unwatch()` method is used to disable the alert when the `href` property has been changed.

Listing 8.305 Using the `unwatch()` Method for the `Link` Object

```

<html>
<script type="text/javascript" a language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{

```

Listing 8.305 Continued

```
document.writeln(id + " changed from " + "<b>" + oldValue + "</b>" +
➤ " to " + "<b>" + newValue + "</b><br>")
return newValue;
}

myLink = document.links.href;
watch("myLink", alertme);
myLink = "http://www.bmw.com";
unwatch("myLink");
myLink = "http://www.amazon.com";
watch("myLink", alertme);
myLink = "http://www.buy.com";

// -->
</script>

<body>
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
➤ Link to Something</a>
</form>

</body>
</html>
```

Link.watch()

JavaScript 1.2+
Nav4+

Syntax

watch(prop, handler)

Description

The watch() method of the Link object is used to watch for changes to Link properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.306 shows how the watch() method is used to determine when the href property of the link has changed.

Listing 8.306 Using the watch() Method on the Link Object

```
<html>
<script type="text/javascript" language="JavaScript">
```

```

<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + "</b>" + " to " +
    ➤ "<b>" + newValue + "</b><br>")
    return newValue;
}

myLink = document.links.href;
watch("myLink",alertme);
myLink = "http://www.bmw.com";
myLink = "http://www.amazon.com";

// -->
</script>

<body>
<form name="form1">
<a href="http://www.sampublishing.com:80/foo?something#foobar">
➤ Link to Something</a>
</form>

</body>
</html>

```

Location

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Location object represents the current Web address displayed in the browser. The Location object has specific properties and methods associated with it, as shown in Table 8.21.

Table 8.21 *Properties and Methods of the Location Object*

Type	Item	Description
Property	hash	Represents an anchor name in the URL that begins with the # character
	host	Represents the hostname and port number of the URL
	hostname	Represents the hostname part of the URL

Table 8.21 Continued

Type	Item	Description
Method	href	Represents the complete URL
	pathname	Represents the pathname part of the URL
	port	Represents the port part of the URL
	protocol	Represents the protocol part of the URL
	search	The search part of the URL, including the ?
	reload()	Reloads the current URL
	replace()	Loads a new Web page in the current browser
	unwatch()	Removes a watchpoint on the Location property
	watch()	Sets a watchpoint on the Location property

Example

Listing 8.307 shows how to access the Location object.

Listing 8.307 Example of the Location Object

```
<html>

<body>

<form name="form1">
Click the button to get the current location value.
<br><br><br>
<input type="button" name="getLoc" value="Get Location"
onClick='alert("The current location is: " + document.location)'\>
<br>
</form>

</body>
</html>
```

Location.hash

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.hash

Description

The hash property of the Location object refers to the anchor portion of the URL, including the hash symbol (#).

In the following fictitious Web address:

`http://www.sampublishing.com:80/foo?something#foobar`

the hash value would be the `#foobar` portion.

Example

Listing 8.308 shows an example of how to get the hash value.

Listing 8.308 *Example of the* hash *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the anchor portion of the URL
function show(){
    document.form1.text1.value=document.location.hash;
}
// -->
</script>

<form name="form1">
Click the button to get the current location.hash
➡ value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.hash value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="getLoc"
➡ value="Get hash value" onClick='show()'>
<br>
</form>

</body>
</html>
```

Location.host

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`location.host`

Description

The host property of the Location object represents the host portion of the URL. This is composed of the hostname and the port number (if available).

In the following fictitious Web address:

```
http://www.sampublishing.com:80/foo?something#foobar
```

the host value would be the `www.sampublishing.com:80` portion.

Example

Listing 8.309 shows the how the host property is used.

Listing 8.309 *Example of the* host *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// shows the host part of the URL
function show(){
    document.form1.text1.value=document.location.host
}
// -->
</script>

<form name="form1">
Click the button to get the current location.host value of
the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.host value: <input type="text" name="text1" size=20>
<br>
<input type="button" name="getLoc" value="Get host" onClick='show()''>
<br>
</form>

</body>
</html>
```

Location.hostname

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.hostname

Description

The hostname property of the Location object represents the hostname portion of the URL.

In the following fictitious Web address:

`http://www.sampublishing.com:80/foo?something#foobar`

the hostname value would be the `www.sampublishing.com` portion.

Example

Listing 8.310 shows an example of how the hostname property is used. When the button is clicked, the `show()` function is called, which displays the value of hostname in the text box.

Listing 8.310 *Example of the* hostname *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays the hostname of the URL
function show(){
    document.form1.text1.value=document.location.hostname;
}
// -->
</script>

<form name="form1">
Click the button to get the current location.hostname
➤value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.hostname value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="gethost"
➤value="Get hostname" onClick='show()'
```

Listing 8.310 Continued

```
<br>
</form>

</body>
</html>
```

Location.href

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.href

Description

The href property of the Location object represents the entire URL string for the current page displayed in the browser.

In the following fictitious Web address:

`http://www.sampublishing.com:80/foo?something#foobar`

the href value would be the entire URL address.

Example

An example of how to get the href value is shown in Listing 8.311.

Listing 8.311 Example of the href Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function show(){
    document.location.href ="http://www.sampublishing.com:80/
    ➤foo?something#foobar";
}
// -->
</script>

<form name="form1">
Click the button to set the current location.href
➤value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
```



```

Location.href value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="sethref" value="Set href" onClick='show() '>
<br>
</form>

</body>
</html>

```

Location.pathname

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.pathname

Description

The pathname property of the Location object represents the pathname portion of the URL.

In the following fictitious Web address:

`http://www.sampublishing.com:80/foo?something#foobar`

the pathname value would be the `/foo` portion.

Example

Listing 8.312 shows an example of how to get the pathname.

Listing 8.312 *Example of the* pathname *Property*

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function to display the pathname value
function show(){
    document.form1.text1.value=document.location.pathname;
}
// -->
</script>

<form name="form1">
Click the button to get the current location.pathname
➡value of the following address:

```

Listing 8.312 Continued

```
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.pathname value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="getLoc" value="Get pathname" onClick='show()'>
<br>
</form>

</body>
</html>
```

Location.port

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.port

Description

The port property of the Location object represents the port portion of the URL. This normally follows the hostname, but is not always available.

In the following fictitious Web address:

http://www.sampublishing.com:80/foo?something#foobar

the port value would be the 80 portion.

Example

Listing 8.313 shows how to get the port number.

Listing 8.313 Example of the port Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function to display the port value
function show(){
    document.form1.text1.value=document.location.port;
}
// -->
</script>
```

```

<form name="form1">
Click the button to get the current location.port
↳value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.port value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="getport" value="Get port"
↳onClick='show()'>
<br>
</form>

</body>
</html>

```

Location.protocol

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

location.protocol

Description

The protocol property of the Location object represents the protocol portion of the URL. This is located in the beginning of the URL address (the text before ://).

In the following fictitious Web address:

http://www.sampublishing.com:80/foo?something#foobar

the protocol value would be the http portion.

Example

The protocol property is used in Listing 8.314.

Listing 8.314 *Example of the protocol Property*

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Function to display the protocol value
function show(){
    document.form1.text1.value=document.location.protocol;
}

```

Listing 8.314 Continued

```
// -->
</script>

<form name="form1">
Click the button to get the current location.protocol
➤value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.protocol value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="getproto" value="Get protocol"
➤onClick='show()''>
<br>
</form>

</body>
</html>
```

Location.reload()

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

```
location.reload()
```

Description

The `reload()` method is used to reload the current page displayed in the browser.

Example

Listing 8.315 shows an example of how the `reload()` method is used.

Listing 8.315 Example of the `reload()` Method

```
<html>

<body>

<form name="form1">
Click the button to reload the current page.
<br><br>
Location.hash value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="load" value="Reload page"
➤ onClick='document.location.reload()''>
<br>
</form>
```

```
</body>
</html>
```

Location.replace()

JavaScript 1.1+, JScript 3.0+
Nav3+, IE4+, Opera3+

Syntax

```
location.replace(URL)
```

Description

The `replace()` method is used to load a new page, specified by *URL*, in the current browser window. The new page replaces the previous page's position in the history list.

Example

Listing 8.316 shows the `replace()` method being used to load the new Web site.

Listing 8.316 *Example of the* `replace()` *Method*

```
<html>

<body>

<form name="form1">
Click the button to load the new page: http://www.sampublishing.com
<br>
<br>
<input type="button" name="load" value="Load new page"
onClick='document.location.replace("http://www.sampublishing.com")'>
<br>
</form>

</body>
</html>
```

Location.search

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
location.search
```

Description

The `search` property of the `Location` object represents the query portion of the URL, including the preceding question mark.

In the following fictitious Web address:

`http://www.sampublishing.com:80/foo?something#foobar`

the search value would be the `?something` portion.

Example

Listing 8.317 shows how to use the search property. When the button is clicked, the show function displays the value of the search property in the text box.

Listing 8.317 *Example of the search Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function displays the search property value
function show(){
    document.form1.text1.value=document.location.search;
}
// -->
</script>

<form name="form1">
Click the button to get the current location.search
➡value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.search value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="getsearch" value="Get search"
➡onClick='show()'>
<br>
</form>

</body>
</html>
```

Location.unwatch()

JavaScript 1.2+
Nav4+

Syntax

`location.unwatch(prop)`

Description

The `unwatch()` method of the `Location` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to unwatch.

Example

Listing 8.318 shows how the `unwatch()` method is used to disable the watchpoint on the `location.host` property.

Listing 8.318 *Using the `unwatch()` Method of the `Location` Object*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue
➡ + "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.location.host;
    watch("myProp",alertme);
    myProp = "amazon";
    unwatch("myProp");
    myProp = "google";
}

// -->
</script>
<body>
<form name="form1">
Click the button to get the current location.host
value of the following address:
<br>
http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Location.host value: <input type="text" name="text1" size="20">
<br>
<input type="button" name="gethost" value="set watch" onClick='setWatch()''>
<br>
</form>

</body>
</html>
```

Location.watch()

JavaScript 1.2+

Nav4+

Syntax

`watch(prop, handler)`

Description

The `watch()` method of the `Location` object is used to watch for changes to `Location` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.319 shows how the `watch()` method is used to determine when the port value has been altered.

Listing 8.319 Using the *watch* Method of the *Location* Object.

```
<html>
<head>
<title>Using the watch method of the Location object</title>
<script type="text/javascript" language="JavaScript1.2">
<!--

// Function to change the value of the Location port
function changeValue(){

    origPort = document.location.port;
    newPort = document.form1.myNewPort.value;
    watch( origPort, informChange(newPort) );
    origPort = newPort;

}

// Function to invoke an alert box from the watch method
function informChange(val){
    alert("The Port value has been changed to: " + val);
}

// -->
</script>
</head>
<body>

<form name="form1">
Click the button to get the current location.port
value of the following address:
<br>
```



```

http://www.sampublishing.com:80/foo?something#foobar
<br><br>
Enter New Location.port value: <input type="text" name="myNewPort" size="20">
<br>
<input type="button" name="port" value="Change port" onClick='changeValue()'>
<br>
</form>

</body>
</html>

```

MimeType

JavaScript 1.1+
Nav3+

Syntax

Core client-side JavaScript object.

Description

The MimeType object is a predefined JavaScript object that you access through the mimeTypees array of the Navigator or Plugin object. MIME stands for multipart Internet mail extensions. Table 8.22 shows the different methods and properties of the MimeType object.

Table 8.22 *Properties and Methods of the MimeType Object*

Type	Item	Description
Property	description	Returns a description of a MimeType
	enabledPlugin	Returns a plug-in for a specific MimeType
	suffixes	Returns a file extension for a MimeType
	type	Returns a string representation of a MimeType
Method	unwatch()	Removes a watchpoint on a MimeType property
	watch()	Sets a watchpoint on a MimeType property

Example

Listing 8.320 shows an example of accessing a MimeType through the mimeTypees array. It prints out a few mimeTypees to the browser window.

Listing 8.320 *Example of MimeType*

```

<html>

<body>
<script type="text/javascript" language="JavaScript">
<!--

```

Listing 8.320 Continued

```
// function prints the first three MimeTypes
for (i=0; i < 3; i++) {
    document.writeln(navigator.mimeTypes[i].type);
    document.writeln(navigator.mimeTypes[i].description);
    document.writeln(navigator.mimeTypes[i].suffixes);
    //document.writeln("<br>");
}
// -->
</script>

</body>
</html>
```

MimeType.description

JavaScript 1.1+
Nav3+

Syntax

*mime*type.description

Description

The description property of the MimeType object is used to obtain a description of the data type described by the MimeType object.

Example

Listing 8.321 shows an example of how the description property is used. A for loop is used to output the suffixes of the first three MimeTypes.

Listing 8.321 Accessing the description Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function prints the suffixes for the first three MimeTypes
for (i=0; i < 3; i++) {
    document.write("MimeType description " + i + " : ");
    document.writeln(navigator.mimeTypes[i].description);
    document.write("<br>");
}
// -->
</script>

</body>
</html>
```

MimeType.enabledPlugin

JavaScript 1.1+

Nav3+

Syntax

mimetype.enabledPlugin

Description

The `enabledPlugin` property of the `MimeType` object is used to determine which plugin is configured for a specific MIME type.

Example

Listing 8.322 shows how the `enabledPlugin` property is used. A for loop is used to output the first three `enabledPlugin` values.

Listing 8.322 *Accessing the `enabledPlugin` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// loop through and output enabledPlugin values
for (i=0; i < 3; i++) {
    document.write("MimeType enabledPlugin " + i + " : ");
    document.writeln(navigator.mimeTypes[i].enabledPlugin);
    document.write("<br>");
}
// -->
</script>

</body>
</html>
```

MimeType.suffixes

JavaScript 1.1+

Nav3+

Syntax

mimetype.suffixes

Description

The `suffixes` property of the `MimeType` object is used to obtain a string listing the possible file suffixes or filename extensions for the MIME type.

Example

Listing 8.323 shows an example of how the `suffixes` property is used. A `for` loop is used to output the first three suffix values.

Listing 8.323 *Accessing the* `suffixes` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function prints the suffixes for the first three MimeTypes
for (i=0; i < 3; i++) {
    document.write("MimeType suffix " + i + " : ");
    document.writeln(navigator.mimeTypes[i].suffixes);
    document.write("<br>");
}
// -->
</script>

</body>
</html>
```

MimeType.type

JavaScript 1.1+
Nav3+

Syntax

`mime.type.type`

Description

The `type` property of the `MimeType` object is used to obtain a string specifying the name of the MIME type.

Example

Listing 8.324 shows how the `type` property is used. A `for` loop is used to output the first three MIME types.

Listing 8.324 *Example of the* `type` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// function prints the types for the first 3 MimeTypes
for (i=0; i < 3; i++) {
    document.write("MimeType type " + i + " :");
    document.writeln(navigator.mimeTypes[i].type);
    document.write("<br>");
}
// -->
</script>

</body>
</html>
```

MimeType.unwatch()

JavaScript 1.2+
Nav4+

Syntax

`unwatch(prop)`

Description

The `unwatch()` method of the `MimeType` object is used to remove a watchpoint on a property set by the `watch()` method. It takes a single parameter, *prop*, which specifies the property on which to remove the watchpoint.

Example

All `MimeType` properties are read-only, hence no example is provided.

MimeType.watch()

JavaScript 1.2+
Nav4+

Syntax

`watch(prop, handler)`

Description

The `watch()` method of the `MimeType` object is used to watch for changes to `MimeType` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

All `MimeType` properties are read-only, hence no example is provided.

navigator

JavaScript 1.0+
Nav2+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The navigator object is a built-in object that is used to obtain information related to the Navigator browser. Table 8.23 shows the different methods and properties of the navigator object.

Table 8.23 *Properties and Methods of the navigator Object*

Type	Item	Description
Property	appName	Refers to the official browser name
	appVersion	Refers to the version information of the browser
	language	Refers to the language of the browser
	mimeTypes	Refers to an array of <code>MimeType</code> objects that contains all the MIME types that the browser supports
	platform	A string representing the platform on which the browser is running
	plugins	Refers to an array of <code>Plugin</code> objects that contains all the plug-ins installed in the browser
	userAgent	String that represents the user-agent header
	javaEnabled()	Function that tests to see that Java is supported in the browser
	plugins.refresh()	Checks for any newly installed plug-ins
	preference()	Allows reading and setting of various user preferences in the browser
Method	taintEnabled()	Tests to see whether data-tainting is enabled
	unwatch()	Removes a watchpoint on a navigator property
	watch()	Sets a watchpoint on a navigator property

Example

Listing 8.325 shows an example for the navigator object.

Listing 8.325 Example of the navigator Object

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Output the navigator appName property
document.write(navigator.appName);
// -->
</script>

</body>
</html>
```

navigator.appCodeName

JavaScript 1.0+
Nav2+, Opera3+

Syntax

navigator.appCodeName

Description

The appCodeName property of the navigator object refers to the internal code name of the browser.

Example

Listing 8.326 shows an example of how the appCodeName property is used.

Listing 8.326 Example of the appCodeName Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// Output the navigate appCodeName property
document.write(navigator.appCodeName);
// -->
</script>

</body>
</html>
```

navigator.appName

JavaScript 1.0+
Nav2+, Opera3+

Syntax

`navigator.appName`

Description

The `appName` property of the `navigator` object refers to the browser name.

Example

Listing 8.327 shows an example of how the `appName` property is used.

Listing 8.327 Example of the `appName` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// output the appName property
document.write(navigator.appName);
// -->
</script>

</body>
</html>
```

navigator.appVersion

JavaScript 1.0+
Nav2+, Opera3+

Syntax

`navigator.appVersion`

Description

The `appVersion` property of the `navigator` object is used to get the browser version. The returned property contains the browser version, platform on which the browser is running, and the country (either international or domestic).

Example

Listing 8.328 shows an example of how the `appVersion` property is used.

Listing 8.328 Example of the `appVersion` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// output the appVersion property
document.write(navigator.appVersion);
// -->
</script>

</body>
</html>
```

`navigator.javaEnabled()`

JavaScript 1.1+
Nav3+, Opera3+

Syntax

```
navigator.javaEnabled()
```

Description

The `javaEnabled()` method is used to test whether the browser supports Java.

Example

Listing 8.329 shows an example of how the `javaEnabled()` method is used. A check is made to determine whether Java is enabled on the browser. If so, a message is output indicating that to be the case.

Listing 8.329 Example of the `javaEnabled()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// check to determine if Java is enabled on the browser.
// If so then output a message.
if (navigator.javaEnabled()){
    document.write("This browser supports Java");
}
```

Listing 8.329 Continued

```
// -->
</script>

</body>
</html>
```

navigator.language

JavaScript 1.2+
Nav4+

Syntax

`navigator.language`

Description

The `language` property of the `navigator` object is used to determine what language the browser supports. The return value is a string.

Example

Listing 8.330 shows an example of how the `language` property is used.

Listing 8.330 Example of the `language` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// output the navigator language property
    document.write(navigator.language);
// -->
</script>

</body>
</html>
```

navigator.mimeTypes

JavaScript 1.1+
Nav3+, Opera3+

Syntax

`navigator.mimeTypes`

Description

The `mimeType` property of the navigator object is used to obtain a list of all the MIME types supported by the browser. The returned object is an array containing all supported MIME types. The `mimetypes` array contains all the properties of the Array object.

Example

Listing 8.331 shows an example of how the `mimeType` property is used.

Listing 8.331 *Example of the `mimeType` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// output the navigator mimeType length property
    document.write(navigator.mimeType.length);
// -->
</script>

</body>
</html>
```

navigator.platform

JavaScript 1.2+
Nav4+

Syntax

`navigator.platform`

Description

The `platform` property of the navigator object returns a string representing the platform on which the browser is running. Valid platform values are: Win32, Win16, Mac68k, MacPPC.

Example

Listing 8.332 shows an example of how the `platform` property is used.

Listing 8.332 *Example of the `platform` Property*

```
<html>

<body>
```

Listing 8.332 Continued

```
<script type="text/javascript" language="JavaScript">
<!--
// output the navigator platform property
    document.write(navigator.platform);
// -->
</script>

</body>
</html>
```

navigator.plugins

JavaScript 1.1+
Nav3+

Syntax

`navigator.plugins[num]`

Description

The `plugins` property of the `navigator` object returns an array of the `Plugins` object representing all the plug-ins installed on a particular browser. These can be accessed by the indexed *num* passed.

Example

Listing 8.333 shows an example of how the `plugins` property is used. A loop is used to output a list of all the plug-ins for the browser.

Listing 8.333 Example of the `plugins` Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
var plugLength = navigator.plugins.length

// loop through and output all the plugins
// present in the browser
for(i=0; i<plugLength; i++){
    document.write(navigator.plugins[i].name);
}
// -->
</script>

</body>
</html>
```

navigator.plugins.refresh()

JavaScript 1.1+

Nav3+

Syntax

`navigator.plugins.refresh()`

Description

The `plugins.refresh()` method of the `navigator` object is used to check for any new plug-ins installed on the browser.

Example

Listing 8.334 shows an example of how the `plugins.refresh()` method is used. When the document is loaded, the `plugins.refresh()` method.

Listing 8.334 *Example of the* `plugins.refresh()` *Method*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
    document.write(navigator.plugins.refresh());
// -->
</script>

</body>
</html>
```

navigator.preference()

JavaScript 1.2+

Nav4+, Opera3+

Syntax

`navigator.preference(name)`

`navigator.preference(name, value)`

Description

The `preference()` method of the `navigator` object is used to read or set any user preferences in the browser. For security reasons, reading a preference with the `preference()` method requires the `UniversalPreferencesRead` privilege. Setting a preference with this method requires the `UniversalPreferencesWrite` privilege.

Example

Listing 8.335 shows an example of how the `preference()` method can be used to set a preference.

Listing 8.335 *Example of the* `preference()` **Method**

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// This disables Java support in the browser
navigator.preference(security.enable_java, false);
// -->
</script>

</body>
</html>
```

navigator.taintEnabled()

JavaScript 1.1+

Nav3

Syntax

```
navigator.taintEnabled()
```

Description

The `taintEnabled()` method of the navigator specifies whether data tainting is enabled.

NOTE

This method is removed in Navigator 4 and later releases.

Example

Listing 8.336 shows an example of how the `taintEnabled()` method is used.

Listing 8.336 *Example of the* `taintEnabled()` **Method**

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// determine if tainting is enabled in the browser
if(navigator.taintEnabled()){
    document.write("Data tainting is enabled");
}
else{
    document.write("There is no data tainting");
}
// -->
</script>

</body>
</html>
```

navigator.userAgent

JavaScript 1.0+
Nav2+, Opera3+

Syntax

navigator.userAgent

Description

The userAgent property of the navigator object returns a string identifying the browser. This value is often sent to servers during HTTP data requests.

Example

Listing 8.337 shows an example of how the userAgent property is used.

Listing 8.337 Example of the userAgent Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
//Returns the userAgent property of the browser

document.write(navigator.userAgent);
// -->
</script>

</body>
</html>
```

navigator.unwatch()

JavaScript 1.2+
Nav4+

Syntax

`unwatch(prop)`

Description

The `unwatch()` method of the `navigator` object is used to remove a watchpoint on a property set by the `watch()` method. It takes a single parameter, *prop*, which specifies the property on which to remove the watchpoint.

Example

All navigator properties are read-only, hence no example is provided.

navigator.watch()

JavaScript 1.2+
Nav4+

Syntax

`watch(prop, handler)`

Description

The `watch()` method of the `navigator` object is used to watch for changes to navigator properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

All navigator properties are read-only, hence no example is provided.

Option

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

Core client-side JavaScript object.

Description

The `Option` object is used to reference a property of an HTML select list. An `Option` object can be created by using the `option` constructor and specifying the appropriate values. Table 8.24 shows the different properties and methods of the `Option` object.

Table 8.24 *Properties and Methods of the Option Object*

Type	Item	Description
Property	defaultSelected	Refers to the option that is selected by default from the select box
	selected	Refers to the selected value of the select box
	text	Refers to the text for the option
	value	Refers to the value that is returned to when the option is selected
Method	unwatch()	Removes a watchpoint on an Option property
	watch()	Sets a watchpoint on an Option property

Example

Listing 8.338 shows an example of how a select list is populated.

Listing 8.338 Example of the Option Object

```
<html>

<body>

<script
<script type="text/javascript" language="JavaScript">
<!--
// function adds the options to the select list.
function add(myForm) {

    var option0 = new Option("Joe", "person1");
    var option1 = new Option("Jane", "person2");
    var option2 = new Option("Mark", "person3");
    var option3 = new Option("Melissa", "person4");

    for (var i=0; i < 4; i++) {
        eval("myForm.people.options[i]=option" + i)
    }

    history.go(0)
}
// -->
</script>

<form>
<select name="people" multiple></select><p>
<input type="button" value="Add People" onClick="add(this.form)">
</form>
</body></html>
```

Option.defaultSelected

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+

Syntax

`option.defaultSelected`

Description

The `defaultSelected` property of the `Option` object specifies the initial option selection of the select list.

Example

Listing 8.339 shows an example of how the `defaultSelected` property is used. The check function checks for the default selected option.

Listing 8.339 *Example of the* `defaultSelected` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function checks the form to see what is the default selected.
function check(myForm){
    for (var i = 0; i < document.form1.myList.length; i++) {
        if (document.form1.myList.options[i].defaultSelected == true) {
            alert("The default value is: 2");
        }
    }
}
// -->
</script>

<form name=form1>
<select name="myList" multiple>
<option value=1>One
<option value=2 selected>Two
<option value=3>Three
<option value=4>Four
</select><p>
<input type="button" value="Get Default Value" onClick='check(this.form)'\>
</form>

</body>
</html>
```

Option.selected

JavaScript 1.2+

Nav4+

Syntax

`option.selected`

Description

The `selected` property of the `Option` object specifies the current selected option of the select list.

Example

Listing 8.340 shows an example of how the `selected` property is used. The check function checks to see what the `selected` property is.

Listing 8.340 *Example of the* `selected` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function checks the forms to see what the selected property is
function check(myForm){
    for (var i = 0; i < document.form1.myList.length; i++) {
        if (document.form1.myList.options[i].selected == true) {
            alert("The selected value is: "
➤+ document.form1.myList.options[i].value);
        }
    }
}
// -->
</script>

<form name=form1>
<select name="myList" multiple>
<option value=1>One
<option value=2>Two
<option value=3>Three
<option value=4>Four
</select><p>
<input type="button" value="Get Value" onClick='check(this.form) '>
</form>

</body>
</html>
```

Option.text

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

option.text

Description

The text property of the Option object specifies the text for the selected option of the select list.

Example

Listing 8.341 shows an example of how the text property is used. The check function determines what the text is for the selected value.

Listing 8.341 *Example of the text Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function check the form to see what the selected value's text is
function check(myForm){
    for (var i = 0; i < document.form1.myList.length; i++) {
        if (document.form1.myList.options[i].selected == true) {
            alert("The selected text value is: "
➤+ document.form1.myList.options[i].text);
        }
    }
}
// -->
</script>

<form name=form1>
<select name="myList" multiple>
<option value=1>One
<option value=2>Two
<option value=3>Three
<option value=4>Four
</select><p>
<input type="button" value="Get Text Value" onClick='check(this.form)'\>
</form>

</body>
</html>
```

Option.unwatch()

JavaScript 1.2+

Nav4+

Syntax

```
option.unwatch(prop)
```

Description

The `unwatch()` method of the `Option` object will remove a watchpoint on a property, `prop`, set by the `watch()` method.

Example

Listing 8.342 shows how the `unwatch()` method is used to remove the watchpoint set on the `Option` value. When the submit button is clicked, the `setWatch()` method is called, which sets the watchpoint and then changes the `Option` value. This causes the `alertme` function to be executed, which displays the results of the change to the browser.

Listing 8.342 Using the `unwatch()` Method of `Option`

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " </b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form.myList.value;
    watch("myProp",alertme);
    myProp = 6;
    myProp = 9;
}

// -->
</script>

<body>

<form name="form">
<select name="myList" multiple>
<option value=1>One
<option value=2 selected>Two
```

Listing 8.342 Continued

```
<option value=3>Three
</select>
<br>

<input type="button" value="submit" onClick='setWatch()'>
</form>

</body>
</html>
```

Option.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

`option.value`

Description

The `value` property of the `Option` object specifies the value that is returned to the server when the option is selected and the form is submitted.

Example

Listing 8.343 shows an example of how the `value` property is used. The function `getValue` gets the value of the selected option.

Listing 8.343 Example of the `value` Property

```
<html>

<body>

<script
<script type="text/javascript" language="JavaScript">
<!--
// Function gets the value of the selected option
function getValue(myForm){
    for (var i = 0; i < document.form1.myList.length; i++) {
        if (document.form1.myList.options[i].selected == true) {
            alert("The selected value is: "
➡+ document.form1.myList.options[i].value);
        }
    }
}
// -->
</script>
```

```

<form name="form1">
<select name="myList" multiple>
<option value=1>One
<option value=2>Two
<option value=3>Three
<option value=4>Four
</select><p>
<input type="button" value="Get Value" onClick='getValue(this.form)'\>
</form>

</body>
</html>

```

Option.watch()

JavaScript 1.2+
Nav4+

Syntax

watch(prop, handler)

Description

The watch() method of the Option object is used to watch for changes to Option properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.344 shows how the watch() method is used to monitor when the value property of the Option object is modified.

Listing 8.344 Using the watch() method of the Option Object

```

<html>
<head>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
➡ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myProp = document.form.myList.value;
    watch("myProp",alertme);
}

```

Listing 8.344 Continued

```
    myProp = 6;
    myProp = 9;
}
// -->
</script>

</head>
<body>

<form name="form">
<select name="myList" multiple>
<option value=1>One
<option value=2 selected>Two
<option value=3>Three
</select>
<br>

<input type="button" value="submit" onClick='setWatch()'a
```

Password

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Password object refers to the HTML element created with the <input> tag, setting the type to specify password. It operates similarly to a standard HTML text input box because input is entered into the box. Instead of displaying the input in cleartext, input is displayed using the * symbol. Table 8.25 shows the different methods and properties of the Password object.

Table 8.25 Properties, Methods and Events of the Password Object

Type	Item	Description
Property	defaultValue	Refers to the value attribute of the HTML password box
	form	Refers to the form that contains the password box
	name	Refers to the name attribute of the HTML password box
	type	Refers to the type attribute of the HTML password box
	value	Refers to the current contents of the password box
Method	blur()	Removes focus from the password box

Type	Item	Description
Event	<code>focus()</code>	Sets focus to the password box
	<code>handleEvent()</code>	Invokes the event handler
	<code>select()</code>	Selects the text entered in the password box
	<code>unwatch()</code>	Removes a watchpoint on a Password property
	<code>watch()</code>	Sets a watchpoint on a Password property
	<code>onBlur</code>	Event handler used when the focus is removed from the password box
	<code>onFocus</code>	Event handler used when the focus is put on the password box

Example

Listing 8.345 shows an example of how the Password object is used to read the value entered in a password box.

Listing 8.345 Example of the Password Object

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Show Password"
➤ onClick='alert(document.form1.pass.value) ' >
</form>

</body>
</html>
```

Password.blur()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
password.blur()
```

Description

The `blur` method for the Password object is used to remove the focus from the password box.

Example

Listing 8.346 shows an example of how the focus is removed from the password box.

Listing 8.346 Example of the `blur()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function that shifts focus when the button is clicked
function shift(){
    document.form1.pass.blur();
    document.form1.txt.value="Get Focus";
}
// -->
</script>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="Text" name="txt" size="10">
<br>
<input type="BUTTON" value="Show Password" onClick='shift() '>
</form>

</body>
</html>
```

Password.defaultValue

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+

Syntax

`password.defaultValue`

Description

The `defaultValue` property of the `Password` object is used to get the HTML value attribute of the password box.

Example

Listing 8.347 shows an example of how the `defaultValue` is used to get the password value.

Listing 8.347 Example of the `defaultValue` Property

```
<html>

<body>

<form name="form1">
```

```
<input type="PASSWORD" Name="pass" size="10" value="pass123">
<br>
<input type="BUTTON" value="Show Password"
  ➤ onClick='alert(document.form1.pass.defaultValue) '>
</form>

</body>
</html>
```

Password.focus()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`password.focus()`

Description

The `focus()` method of the Password object sets the focus to the password box.

Example

Listing 8.348 shows an example of how the `focus()` method is used to move the focus back to the password box.

Listing 8.348 Example of the `focus()` Method

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Show Password"
  ➤ onClick='document.form1.pass.focus()'>
</form>

</body>
</html>
```

Password.form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`password.form`

Description

The `form` property of the `Password` object is used to access the `Form` object in which the password box resides.

Example

Listing 8.349 shows an example of how the `form` property is used to get the form name.

Listing 8.349 *Example of the* `form` *Property*

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Show Formname"
  ⚡ onClick='alert(document.form1.pass.form.name) ' >
</form>

</body>
</html>
```

Password.handleEvent()

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+, Opera3+

Syntax

```
password.handleEvent(event)
```

Description

The `handleEvent()` method of the `Password` object is used to handle events of the type of *event* passed.

Example

Listing 8.350 shows an example of how the `handleEvent()` method is used to handle the `Click` event when the button is clicked.

Listing 8.350 *Example of the* `handleEvent()` *Method*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
```

```
// sets up the document to capture CHANGE events
document.captureEvents(Event.CHANGE);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handleChange(evnt){
    window.document.pass.handleEvent(evnt);
}

function showMsg(){
    alert("Password Changed.");
}

// This registers the handle function as the event handler for the
// CHANGE event.
document.onChange = handleChange;
// -->
</script>

<form name="form1">

Enter password:
<input type="PASSWORD" name="pass" size="15" onChange='showMsg()'>
</form>

</body>
</html>
```

Password.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

password.name

Description

The name property of the Password object is used to get the HTML name attribute of the password box.

Example

Listing 8.351 shows an example of how the name property is used. When the button is clicked, an alert box is displayed showing the password object name.

Listing 8.351 Example of the `name` Property

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Show Formname"
➡ onClick='alert(document.form1.pass.name)'\>
</form>

</body>
</html>
```

Password.onBlur

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onBlur="command"
```

Description

The `onBlur` event handler of the `Password` object is used to handle the event that occurs when the focus is removed from the password box.

Example

Listing 8.352 shows an example of how the `onBlur` event handler is used. When the focus is removed from the `Password` object, the `setTxt()` function is called to display a message in the text box.

Listing 8.352 Example of the `onBlur` Event Handler

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function that sets the text value
function setTxt(){
    document.form1.txt.value="setup";
}
// -->
</script>
```

```

<form name="form1">
<input type="PASSWORD" Name="pass" size="10" onBlur='setTxt()'>
<br>
<input type="Text" name="txt" size="10" >
</form>

</body>
</html>

```

Password.onFocus

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

onFocus="command"

Description

The onFocus event handler of the Password object is used to handle the Focus event for the password box.

Example

Listing 8.353 shows an example of the onFocus event handler.

Listing 8.353 Example of the onFocus Event Handler

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets a message in the text box
function set(){
    document.form1.pass.value="aha123";
}
// -->
</script>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10" onFocus='set()'>
<br>
<input type="BUTTON" value="Show Formname" >
</form>

</body>
</html>

```

Password.select()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
password.select()
```

Description

The `select()` method of the Password object is used to select the value entered into the password box. The selected value is highlighted.

Example

Listing 8.354 shows an example of how the password value can be selected.

Listing 8.354 *Example of the* `select()` *Method*

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Select Password"
  ⚡onClick='document.form1.pass.select()'>
</form>

</body>
</html>
```

Password.type

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
password.type
```

Description

The `type` property of the Password object is used to get the HTML type attribute associated with the password box. For the Password object, this value is always `password`.

Example

Listing 8.355 shows an example of how the `type` property is used. When the button is clicked, an alert box is displayed, showing the value of the `type` property.

Listing 8.355 Example of the `type` Property

```

<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Get Type"
➡ onClick='alert(document.form1.pass.type)'\>
</form>

</body>
</html>

```

Password.unwatch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

```
password.unwatch(prop)
```

Description

The `unwatch()` method of the `Password` object will remove a watchpoint on a property, `prop`, set by the `watch()` method.

Example

Listing 8.356 shows how the `unwatch()` method is used to turn off the watchpoint set on the `Password` value.

Listing 8.356 Using the `unwatch()` Method for `Password`

```

<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
  document.writeln(id + " changed from " + "<b>" + oldValue +
➡ "</b>" + " to " + "<b>" + newValue + "</b><br>")
  return newValue;
}

function setWatch(){
  myPass = document.form1.pass.value;
  watch("myPass",alertme);
  myPass = "abc123";

```

Listing 8.356 Continued

```
    unwatch("myPass");
    myPass = "purejs2";
}

// -->
</script>

<form name="form1">
Enter Password <input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Submit" onClick='setWatch()'>
</form>

</body>
</html>
```

Password.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

password.value

Description

The value property of the Password object is used to get the value entered in the password box.

Example

Listing 8.357 shows an example of how the value property is used.

Listing 8.357 Example of the value Property

```
<html>

<body>

<form name="form1">
<input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Get Value"
➡ onClick='alert(document.form1.pass.value)'\>
</form>

</body>
</html>
```

Password.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`watch(prop, handler)`

Description

The `watch()` method of the `Password` object is used to watch for changes to `Password` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.358 shows how the `watch()` method is used to inform the user that the password value has changed within the `changeValue()` function.

Listing 8.358 Using the `watch()` Method on the `Password` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + "</b>" +
    " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myPass = document.form1.pass.value;
    watch("myPass",alertme);
    myPass = "abc123";
}

// -->
</script>

<form name="form1">
Enter Password <input type="PASSWORD" Name="pass" size="10">
<br>
<input type="BUTTON" value="Submit" onClick='setWatch()'>
</form>

</body>
</html>
```

Plugin

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The `Plugin` object is used to obtain plug-in information from the browser. The `Plugin` object contains an array of elements containing the MIME types handled by each plug-in. Table 8.26 shows the different properties and methods of the `Plugin` object.

Table 8.26 *Properties and Methods of the `Plugin` Object*

Type	Item	Description
Property	<code>description</code>	Refers to a description of the plug-in
	<code>filename</code>	Refers to the filename of a plug-in program
	<code>length</code>	Refers to the number of MIME types supported
	<code>name</code>	Refers to the plug-in name
Method	<code>unwatch()</code>	Removes a watchpoint on a <code>Plugin</code> property
	<code>watch()</code>	Sets a watchpoint on a <code>Plugin</code> property

Example

Listing 8.359 shows an example in which all the browser's plug-ins are printed to the document.

Listing 8.359 *Example Using the `Plugin` Object*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// function that prints all the browser plug-ins
for(i=0; i<navigator.plugins.length; i++){
    document.write(navigator.plugins[i].description);
}
// -->
</script>

</body>
</html>
```

Plugin.description

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

plugin.description

Description

The `description` property of the `Plugin` object is used to obtain a description of the browser plug-ins.

Example

Listing 8.360 shows an example of how the `description` property is used.

Listing 8.360 *Example of the* `description` *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// print the description for the first plug-in
    document.write(navigator.plugins[0].description);

// -->
</script>

</body>
</html> >
```

Plugin.filename

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

plugin.filename

Description

The `filename` property of the `Plugin` object is used to get the path and filename for a plug-in.

Example

Listing 8.361 shows an example of how the `filename` property is used.

Listing 8.361 Example of the filename Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// print the filename for the first browser plug-in
    document.write(navigator.plugins[0].filename);

// -->
</script>

</body>
</html>
```

Plugin.length

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

plugin.length

Description

The length property of the Plugin object determines the number of MIME data types the plug-in can support.

Example

Listing 8.362 shows an example of how the length property is used.

Listing 8.362 Example of the length Property

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// print the number of browser plug-ins
    document.write(navigator.plugins.length);

// -->
</script>

</body>
</html>
```

Plugin.name

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

plugin.name

Description

The name property of the Plugin object is used to get the plug-in's name.

Example

Listing 8.363 shows an example of how the name property is used.

Listing 8.363 *Example of the name Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// print the name of the first browser plug-in
    document.write(navigator.plugins[0].name);

// -->
</script>

</body>
</html>
```

Plugin.unwatch()

JavaScript 1.2+
Nav4+

Syntax

unwatch(prop)

Description

The *unwatch()* method of the Plugin object is used to disable a watch on a Plugin property. Specify the property, *prop*, to disable.

Example

All Plugin properties are read-only, hence no example is provided.

Plugin.watch()

JavaScript 1.2+
Nav4+

Syntax

watch(prop, handler)

Description

The watch() method of the Plugin object is used to watch for changes to Plugin properties. When one of the properties, prop, is assigned a value, a handler is used to call a user-defined function.

Example

All Plugin properties are read-only, hence no example is provided.

Radio

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Radio object represents a check box within an HTML form. A check box is created using the HTML <input> tag and specifying the type attribute as radio. The Radio object has specific properties and methods associated with it, as shown in Table 8.27.

Table 8.27 Properties, Methods, and Events of the Radio Object

Type	Item	Description
Property	checked	Specifies whether a button is checked or unchecked
	defaultChecked	Refers to the checked attribute of the HTML <input> tag
	form	Refers to the Form object that contains the Radio object
	name	Refers to the name attribute of the HTML <input> tag
	type	Refers to the type attribute of the HTML <input> tag
	value	Refers to the value attribute of the HTML <input> tag
Method	blur()	Removes focus from the Radio object
	click()	Simulates a mouse click on the button

Type	Item	Description
Event	<code>focus()</code>	Sets the focus to a button
	<code>handleEvent()</code>	Invokes the default handler for the specified event
	<code>unwatch()</code>	Removes a watchpoint on a <code>Radio</code> property
	<code>watch()</code>	Sets a watchpoint on a <code>Radio</code> property
	<code>onBlur</code>	Event handler for the <code>Blur</code> event
	<code>onClick</code>	Event handler for the <code>Click</code> event
	<code>onFocus</code>	Event handler for the <code>Focus</code> event

Example

Listing 8.364 shows how a check box is created and then how the `name` property is accessed using the `Radio` object.

Listing 8.364 Example of the `Radio` Object

```
<html>

<body>

<form name="form1">
<input type="radio" name=button1
☛onClick='alert(document.form1.button1.name)'+>Box 1
<br>
<input type="radio" name="button2">Box 2
<br>
</form>

</body>
</html>
```

Radio.blur()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
radio.blur()
```

Description

The `blur()` method of the `Radio` object is used to remove the focus from the check box.

Example

Listing 8.365 shows how the `blur()` method is used to remove focus from the radio button. When the Remove Focus button is clicked, the focus is removed from the radio button and a text message is displayed.

Listing 8.365 *Example of the `blur()` Method*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function removes focus from the button
function change(){
document.form1.button1.blur();
document.form1.text1.value="Focus removed from button";
}
// -->
</script>

<form name="form1">
<input type="radio" name="button1" CHECKED>Box 1
<br>
<input type="radio" name="button2">Box 2
<br>
<input type="button" value="Remove Focus" onClick='change()'>
<br>
<input type="text" name="text1" size="15">
</form>

</body>
</html>
```

Radio.checked

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`radio.checked`

Description

The `checked` property of the `Radio` object is a boolean value used to determine whether a radio button is in a checked or unchecked state.

Example

Listing 8.366 shows an example of how to determine whether the radio button is checked by using the checked property.

Listing 8.366 *Example of the* checked *Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function looks to see if button 1 is checked and displays
// an appropriate alert message.
function checkButton(){
if(document.form1.button1.checked == true){
    alert("Box1 is checked");
}
else if(document.form1.button2.checked == true){
    alert("Box 2 is checked");
}
}
// -->
</script>

<form name="form1">
<input type="radio" name="button1">Box 1
<br>
<input type="radio" name="button2" CHECKED>Box 2
<br>
< INPUT type="button" value="Get Checked" onClick='checkButton()'>
</form>

</body>
</html>
```

Radio.click()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
radio.click()
```

Description

The click() property of the Radio object is used to simulate a mouse click on the radio button.

Example

Listing 8.367 shows an example of how the `click()` property is used.

Listing 8.367 *Example of the `click()` Property*

```
<html>

<body>

By pressing the "Simulate Click" button below, box 2 becomes checked
because a simulated mouse click is performed.
<form name="form1">
<input type="radio" name="button1" CHECKED>Box 1
<br>
<input type="radio" name="button2">Box 2
<br>
<input type="button" value="Simulate Click"
  ➤ onClick='document.form1.button2.click()'>
</form>

</body>
</html>
```

Radio.defaultChecked

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`radio.defaultChecked`

Description

The `defaultChecked` property of the `Radio` object is a boolean value that reports which radio buttons contain the HTML `checked` attribute. If the `checked` attribute is contained in the `Radio` object, `true` is returned. Otherwise, `false` is returned.

Example

Listing 8.368 shows how the `defaultChecked` property is used to find out which box is checked by default.

Listing 8.368 *Example of the `defaultChecked` Property*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function checkBox(){
```

```

if(document.form1.button1.defaultChecked == true){
    alert("Box1 is checked by default");
}
else if(document.form1.button2.defaultChecked == true){
    alert("Box 2 is checked by default");
}
}
}
// -->
</script>

<form name="form1">
<input type="radio" name="button1">Box 1
<br>
<input type="radio" name="button2" CHECKED>Box 2
<br><br>
<input type="button" value="Get Default" onClick='checkBox()'>
</form>

</body>
</html>

```

Radio.focus()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
radio.focus()
```

Description

The `focus()` method of the `Radio` object sets the focus to the radio button.

Example

Listing 8.369 shows how the `focus()` method is used.

Listing 8.369 Example of the `focus()` Method

```

<html>

<body>

<form name="form1">
<input type="radio" name="button1">Box 1
<br>
<input type="radio" name="button2">Box 2
<br><br>
<input type="button" value="Set Focus to Box 1"
➤ onClick='document.form1.button1.focus()>

```

Listing 8.369 Continued

```
</form>

</body>
</html>
```

Radio.form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

radio.form

Description

The form property of the Radio object is used to reference the Form object that contains the Radio box.

Example

Listing 8.370 shows an example of how the form property can be used to get the name of the form that contains the Radio object.

Listing 8.370 Example of the form Property

```
<html>

<body>

<form name="form1">
<input type="radio" name="button1">Box 1
<br>
<input type="radio" name="button2">Box 2
<br><br>
<input type="button" value="Get Form Name"
  onClick='alert("The form name is: " + document.form1.button1.form.name)'\>
</form>

</body>
</html>
```

Radio.handleEvent()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

radio.handleEvent(*event*)

Description

The `handleEvent()` method of the `Radio` object invokes the event handler for the specific event.

Example

Listing 8.371 shows how the `handleEvent()` method is used to capture all `CLICK` events and pass them to the `but1` object event handler.

Listing 8.371 *Using the* `handleEvent()` *Method*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets up the document to capture CLICK events
document.captureEvents(Event.CLICK);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handleClick(evnt){
    window.document.but1.handleEvent(evnt);
}

function showMsg(){
    alert("Button clicked.");
}

// This registers the handle function as the event handler for the
// CLICK event.
document.onClick = handleClick;
// -->
</script>

<form name="form1">
<input type="radio" name="but1" onClick='showMsg()'>Choice 1
<br>
<input type="radio" name="but1" onClick='showMsg()'>Choice 2
<br>
</form>

</body>
</html>
```

Radio.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

radio.name

Description

The name property of the Radio object represents the name attribute of the HTML `<input>` tag that creates the Radio button. This allows you to reference a Radio object directly by name.

Example

Listing 8.372 shows how the name of the Radio button is used to access its properties.

Listing 8.372 *Accessing the Radio Button by Name*

```
<html>

<body>

<form name="form1">
<input type="radio" name="myButton">Box 1
<br><br>
<input type="button" value="Get Name"
onClick='alert("The name of the button is: " +
document.form1.myButton.name) ' >

</form>

</body>
</html>
```

Radio.onBlur

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`onBlur="command"`

Description

The `onBlur` property is an event handler for the Radio object that notifies you when the focus is removed from a radio button.

Example

Listing 8.373 shows how the `onBlur` event handler is used to detect when the focus is removed from the specified radio button.

Listing 8.373 Example of the `onBlur` Event Handler

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function showChange(){
    document.form1.text1.value = "Focus removed from Radio Button";
}
// -->
</script>

<form name="form1">
Click the radio button first, then click in the text area.
<br><br>
<input type="radio" name="button1" onBlur='showChange() '>Box 1
<br>
Click the text box
<input type="text" name="text1" size="40">
<br>
</form>

</body>
</html>

```

Radio.onClick

JavaScript 1.0+, JScript 1.0+
 Nav2+, IE3+, Opera3+

Syntax

```
onClick="command"
```

Description

The `onClick` property of the `Radio` object is an event handler that notifies you when the mouse has been clicked on the button.

Example

Listing 8.374 uses the `onClick` event handler to check for a mouse click event.

Listing 8.374 Example of the `onClick` Event Handler

```

<html>

<body>

<form name="form1">

```

Listing 8.374 Continued

```
<input type="radio" name="button1"
  ⚡onClick='alert("This mouse was clicked on this object")'>Box 1
<br>
</form>

</body>
</html>
```

Radio.onFocus()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onFocus="command"
```

Description

The onFocus event handler of the Radio object is an event handler that notifies you when the focus is set on the radio button.

Example

In Listing 8.375, the onFocus event handler notifies the user with a text message when the focus is set on the radio button.

Listing 8.375 Example of the onFocus Event Handler

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// function sets the focus to box 1
function showFocus(){
    document.form1.text1.value = "Focus set on Box 1";
}
// -->
</script>

<form name="form1">
Click on the radio button
<br><br>
<input type="radio" name="button1" onFocus='showFocus()'>Box 1
<br><br>
<input type="text" name="text1" size="40">
```

```
</form>
```

```
</body>
```

```
</html>
```

Radio.type

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

radio.type

Description

The type property of the Radio button represents the button's type HTML attribute. For this object, it is always radio.

Example

Listing 8.376 shows an example of the type property.

Listing 8.376 Example of the type Property

```
<html>
```

```
<body>
```

```
<form name="form1">
```

```
<input type="radio" name="button1">Box 1
```

```
<br><br>
```

```
<input type="button" value="Get Button Type"
```

```
onClick='alert("The button type is: " + document.form1.button1.type)'
```

```
</form>
```

```
</body>
```

```
</html>
```

Radio.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

radio.unwatch(prop)

Description

The unwatch() method of the Radio object will remove a watchpoint on a property, *prop*, set by the watch() method.

Example

Listing 8.377 shows how the `unwatch()` method is used to turn off the watchpoint on the `myRadio` object.

Listing 8.377 *Using the `unwatch()` Method*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
➡ "</b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myRadio = document.form1.button1.value;
    watch("myRadio",alertme);
    myRadio = 4;
    unwatch("myRadio");
    myRadio = 6;
}

// -->
</script>

<body>
<form name="form1">
<input type="radio" name="button1" value="1">Box 1
<br><br>
<input type="button" value="Get Button Value" onClick='setWatch()''>
</form>

</body>
</html>
```

Radio.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`radio.value`

Description

The `value` property of the `Radio` object represents the `value` attribute of the HTML `<input>` tag used to create the radio button.

Example

Listing 8.378 shows how to access the value property.

Listing 8.378 *Example of the value Property*

```
<html>

<body>

<form name="form1">
<input type="radio" name="button1" value="1">Box 1
<br><br>
<input type="button" value="Get Button Value"
onClick='alert("The button value is: " + document.form1.button1.value)'\>
</form>

</body>
</html>
```

Radio.watch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

```
radio.watch(prop, handler)
```

Description

The watch() method of the Radio object is used to watch for changes to Radio properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.379 shows how the watch() method is used to track when the value of the Radio button has been changed.

Listing 8.379 *Using the watch() Method on the Radio Value Property*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
" "</b>" + " to " + "<b>" + newValue + "a")
    return newValue;
}
```

Listing 8.379 Continued

```
function setWatch(){
    myRadio = document.form1.button1.value;
    watch("myRadio",alertme);
    myRadio = "4";
    myRadio = "6";
}

// -->
</script>

<body>
<form name="form1">
<input type="radio" name="button1" value="1">Box 1
<br><br>
<input type="button" value="Get Button Value" onClick='setWatch()''>
</form>

</body>
</html>
```

Reset

JavaScript 1.2+, JScript 1.0+
Nav4+, IE3+

Syntax

Core client-side JavaScript object.

Description

The Reset object represents a Reset button within an HTML form. The button is created using the HTML `<input>` tag and specifying the `type` attribute as `reset`. Reset buttons are used for form submissions. The Reset object has specific properties and methods associated with it, as shown in Table 8.28.

Table 8.28 Properties, Methods, and Events of the Reset Object

Type	Item	Description
Property	form	Specifies the form name that contains the Reset button
	name	HTML name attribute for the Reset button
	type	HTML type attribute for the Reset button
	value	HTML value attribute for the Reset button
Method	blur()	Removes focus from the Reset button
	click()	Simulates a mouse click on a Reset button
	focus()	Sets the focus to the Reset button
	handleEvent()	Invokes the event handler

Type	Item	Description
Event	<code>unwatch()</code>	Removes a watchpoint on a <code>Reset</code> property
	<code>watch()</code>	Sets a watchpoint on a <code>Reset</code> property
	<code>onBlur</code>	Event handler for the <code>Blur</code> event
	<code>onClick</code>	Event handler for the <code>Click</code> event
	<code>onFocus</code>	Event handler for the <code>Focus</code> event

Example

Listing 8.380 shows the syntax for using the `Reset` object.

Listing 8.380 *Example of the* `Reset` *Object*

```
<html>

<body>

<form name="form1">
Click the Reset button to reset the form.
<br><br><br>
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset">
<br>
<br>
</form>

</body>
</html>
```

Reset.blur()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
reset.blur()
```

Description

The `blur()` method of the `Reset` object is used to remove the focus from the `Reset` button.

Example

Listing 8.381 shows an example of how the `blur()` method is used.

Listing 8.381 Example of the `blur()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--

// displays a message when focus is removed from the reset button
function showMsg(){
    document.form1.msg.value="Focus removed from the Reset button.";
}
// -->
</script>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br>
Message: <input type="text" Name="msg" Size="40">
<br>
<input type="reset" name="resetbutton" value="Reset" onBlur='showMsg()'>
<br>
<br>
</form>

</body>
</html>
```

Reset.click()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
reset.click()
```

Description

The `click()` method of the `Reset` object is used to simulate a mouse click on the `Reset` button.

Example

Listing 8.382 shows how the `click()` method is used.

Listing 8.382 Example of the `click()` Method

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function clickReset(){
    document.form1.resetbutton.click();
}

function inform(){
    alert("The Reset button was clicked");
}
// -->
</script>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset" onClick='inform() '>
<br>
<br>
<input type="button" value="Simulate Click" onClick='clickReset() '>
</form>

</body>
</html>

```

Reset.focus()

JavaScript 1.1+, JScript 1.0+
 Nav3+, IE3+, Opera3+

Syntax

```
reset.focus()
```

Description

The `focus()` method of the `Reset` object is used to set the focus to the Reset button.

Example

Listing 8.383 shows an example of how the `focus()` method is used.

Listing 8.383 Accessing the `focus()` Method

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function focusReset(){
    document.form1.resetbutton.focus();
}
// -->
</script>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset">
<br>
<br>
<input type="button" value="Set Focus to Reset" onClick='focusReset() '>
<br>
</form>

</body>
</html>
```

Reset.form

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`reset.form`

Description

The `form` property of the `Reset` object is used to obtain the name of the form that contains the `Reset` button.

Example

Listing 8.384 shows an example of how to get the form name.

Listing 8.384 Example of Using the `form` Property

```
<html>

<body>
```

```

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset"
onClick='alert("The form name is: " + document.form1.resetbutton.form.name)'\>
<br>
<br>
</form>

</body>
</html>

```

Reset.handleEvent()

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

```
reset.handleEvent(event)
```

Description

The `handleEvent()` method of the `Reset` object invokes the event handler for the object.

Example

Listing 8.385 shows an example of how the `handleEvent()` method is used.

Listing 8.385 Example of the `handleEvent()` Method

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
// sets up the document to capture CLICK events
document.captureEvents(Event.CLICK);

// function that handles the specific event. The evnt parameter refers to
// the event object.
function handleMyClick(evnt){
    window.document.resetbutton.handleEvent(evnt);
}

```

Listing 8.385 Continued

```
function showMsg(){
    alert("Form has been reset");
}

// This registers the handle function as the event handler for the
// CLICK event.
document.onClick = handleMyClick;
// -->
</script>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset" onClick='showMsg()'>
<br>
<br>
</form>

</body>
</html>
```

Reset.name

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`reset.name`

Description

The name property of the Reset object represents the name given to a Reset button as specified from the name attribute of the HTML `<input type="reset">` tag.

Example

In Listing 8.386, the name property is used to inform the user of the active frame.

Listing 8.386 Example of the `name` Property

```
<html>

<body>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
```

```

Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset"
onClick='alert("The reset button name is: " +
document.form1.resetbutton.name) ' >
<br>
<br>
</form>

</body>
</html>

```

Reset.onBlur

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

onBlur="command"

Description

The onBlur event handler for the Reset object is an event handler that specifies when the focus has been removed from the Reset button.

Example

In Listing 8.387, the onBlur event handler is used.

Listing 8.387 Using the onBlur Event Handler

```

<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function setMsg(){
    document.form1.text1.value = "Focus was removed from the reset object";
}
// -->
</script>

<form name="form1">
Click the reset button and then click in a text area.
<br><br>
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>

```

Listing 8.387 Continued

```
Message: <input type="text" name="text1" size="45">
<br><br>
<input type="reset" name="resetbutton" value="Reset" onBlur='setMsg()'>
<br>
<br>
</form>

</body>
</html>
```

Reset.onClick

JavaScript 1.1+, JScript 1.0+
Nav3+, IE3+, Opera3+

Syntax

```
onClick="command"
```

Description

The `onClick` property is an event handler used to handle the event of a mouse click on the Reset button.

Example

Listing 8.388 shows an example of how the `onClick` event handler is used.

Listing 8.388 Example of the `onClick` Event Handler

```
<html>

<body>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset"
onClick='alert("The reset button was clicked")'>
<br>
<br>
</form>

</body>
</html>
```

Reset.onFocus

JavaScript 1.2+, ECMAScript 1.0+, JScript 1.0+
Nav4+, IE3+

Syntax

onFocus="command"

Description

The onFocus event handler is used to specify when the focus is set on the Reset button.

Example

Listing 8.389 shows the onFocus event handler being used to inform the user when the focus is set.

Listing 8.389 *Example of the* onFocus *Event Handler*

```
<html>

<body>

<script type="text/javascript" language="JavaScript">
<!--
function setMsg(){
    document.form2.text1.value = "Focus is set on the reset button.";
}
// -->
</script>

<form name="form1">
Click the reset button.
<br><br>
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset" onFocus='setMsg()'>
<br>
</form>

<form name="form2">
<br><br>
Message: <input type="text" name="text1" size="45">
</form>

</body>
</html>
```

Reset.type

JavaScript 1.2+, ECMAScript 1.0+
Nav4+

Syntax

`reset.type`

Description

The type property of the Reset object represents the HTML type attribute of the Reset button. In the case of a Reset button, this will always be reset.

Example

Listing 8.390 shows how to get the Reset button type.

Listing 8.390 *Example of the type Property*

```
<html>

<body>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset"
onClick='alert("The button type is: " + document.form1.resetbutton.type)'\>
<br>
<br>
</form>

</body>
</html>
```

Reset.unwatch()

JavaScript 1.2+
Nav4+, NES3+

Syntax

`reset.unwatch(prop)`

Description

The `unwatch()` method of the `Reset` object is used to remove a watchpoint set on a property by the `watch()` method. It takes one parameter, which is the property to unwatch.

Example

Listing 8.391 shows how the `unwatch()` method to turn off the watch on the `Reset` object value property.

Listing 8.391 *Using the `unwatch()` Method on the `Reset` Object*

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue + 
    " </b>" + " to " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myRes = document.form1.resetbutton.value;
    watch("myRes",alertme);
    myRes = "cat";
    unwatch("myRes");
    myRes = "dog";
}

// -->
</script>

<body>
<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
<input type="reset" name="resetbutton" value="Reset" onClick='setWatch()'>
<br>
<br>
</form>
</body>
</html>
```

Reset.value

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

`reset.value`

Description

The value property of the Reset object represents the HTML value attribute of the Reset button.

Example

Listing 8.392 shows how the Reset button value can be accessed.

Listing 8.392 *Example of the value Property*

```
<html>

<body>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
Enter Phone: <input type="text" Name="phone" Size="10">
<br><br>
<input type="reset" name="resetbutton" value="Reset"
onClick='alert("The button value is: " + document.form1.resetbutton.value) '>
<br>
<br>
</form>

</body>
</html>
```

Reset.watch()

JavaScript 1.2+
 Nav4+, NES3+

Syntax

```
reset.watch(prop, handler)
```

Description

The `watch()` method of the `Reset` object is used to watch for changes to `Reset` properties. When one of the properties, *prop*, is assigned a value, a *handler* is used to call a user-defined function.

Example

Listing 8.393 shows how the `watch()` method is used to watch for a change to the `Reset` object value property. When the value is changed, the *alertme* function is called.

Listing 8.393 Using the `watch()` Method on the `Reset` Object

```
<html>
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id,oldValue,newValue)
{
    document.writeln(id + " changed from " + "<b>" + oldValue +
    " " + "</b>" + " to " + " " + "<b>" + newValue + "</b><br>")
    return newValue;
}

function setWatch(){
    myRes = document.form1.resetbutton.value;
    watch("myRes",alertme);
    myRes = "cat";
    myRes = "dog";
}

// -->
</script>
```

Listing 8.393 Continued

```
<body>

<form name="form1">
Enter Name: <input type="text" Name="name" Size="15">
<br>
<input type="reset" name="resetbutton" value="Reset" onClick='setWatch()'>
<br>
<br>
</form>
</body>
</html>
```

screen

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

Core client-side JavaScript object.

Description

The screen object provides a programmer with access to various properties of the user's screen. These properties are often accessed and used for sizing windows that various scripts open. Table 8.29 shows the properties of the screen object. This object only has two methods, `watch()` and `unwatch()`, which allow you to turn on and off a watch for a particular property. See their respective entries for more information.

Table 8.29 *Properties of the screen Object*

Item	Description
<code>availHeight</code>	Accesses the pixel height of the user's screen, minus the toolbar or any other "permanent" objects.
<code>availLeft</code>	Accesses the x coordinate of the first pixel not part of the user's toolbar or any other "permanent" objects.
<code>availTop</code>	Accesses the y coordinate of the first pixel not part of the user's toolbar or any other "permanent" objects.
<code>availWidth</code>	Accesses the pixel width of the user's screen, minus the toolbar or any other "permanent" objects.
<code>colorDepth</code>	Accesses the maximum number of colors the user's screen can display. This is in bit format.
<code>height</code>	Accesses the pixel height of the user's screen.
<code>pixelDepth</code>	Accesses the number of bits per pixel of the user's screen. Some versions of Internet Explorer do not support this property, which returns undefined.
<code>width</code>	Accesses the pixel width of the user's screen.

Example

Listing 8.394 has a button that, when clicked, opens a secondary window that displays all the properties of the screen object.

Listing 8.394 *Displaying the Properties of the screen Object*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--
```

```
// Define the openWin function called by pressing the button.
function openWin(){
```

Listing 8.394 Continued

```
// Open a window to store the results.
var myWin = open("", "", "width=450,height=200");

// Write the screen properties to the window.
myWin.document.write("The availHeight is: " + screen.availHeight + "<br>");
myWin.document.write("The availWidth is: " + screen.availWidth + "<br>");
myWin.document.write("The colorDepth is: " + screen.colorDepth + "<br>");
myWin.document.write("The height is: " + screen.height + "<br>");
myWin.document.write("The pixelDepth is: " + screen.pixelDepth + "<br>");
myWin.document.write("The width is: " + screen.width + "<br>");

// Close the stream to the window.
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
  <input type="button" value="Click to See Screen Properties" name="myButton"
    ↪onclick="openWin()">
</form>
</body>
</html>
```

screen.availHeight

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

screen.availHeight

Description

The `availHeight` property of the `screen` object accesses the available pixel height of the user's screen. This height is minus any toolbar or any other "permanent" objects that might be on the user's screen.

Example

Listing 8.395 displays the available height of the user's screen to the page.

Listing 8.395 Accessing the `availHeight` Property of the `screen` Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--
```

```
document.write("The available height of this user's screen is <b>");
document.write(screen.availHeight + '</b> pixels');

// -->
</script>
```

screen.availLeft

JavaScript1.2+
Nav4+

Syntax

screen.availLeft

Description

The availLeft property of the screen object specifies the first available x coordinate pixel not allocated to any of the user's toolbar or any other "permanent" objects that may be on the screen.

NOTE

If you access this property in Netscape 6, it does not consider the My Sidebar as part of the window—just the rendering portion window.

Example

Listing 8.396 displays the available left pixel of the user's screen to the page.

Listing 8.396 Accessing the availLeft Property of the screen Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The available left of this user's screen is <b>");
document.write(screen.availLeft + '</b> pixels');

// -->
</script>
```

screen.availTop

JavaScript1.2+
Nav4+

Syntax

screen.availTop

Description

The `availTop` property of the screen object specifies the first available y coordinate pixel not allocated to any of the user's toolbar or any other "permanent" objects that might be on the screen.

Example

Listing 8.397 displays the available top pixel of the user's screen to the page.

Listing 8.397 Accessing the `availTop` Property of the screen Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The available top of this user's screen is <b>");
document.write(screen.availTop + '</b> pixels');

// -->
</script>
```

screen.availWidth

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

`screen.availWidth`

Description

The `availWidth` property of the screen object accesses the available pixel width of the user's screen. This width is minus any toolbar or any other "permanent" objects that may be on the user's screen.

Example

Listing 8.398 displays the available width of the user's screen to the page.

Listing 8.398 Accessing the `availWidth` Property of the screen Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The available width of this user's screen is <b>");
document.write(screen.availWidth + '</b> pixels');

// -->
</script>
```

screen.colorDepth

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

screen.colorDepth

Description

The colorDepth property of the screen object accesses the maximum number of colors the user's screen can display. The returned value is in terms of bits.

Example

Listing 8.399 displays the color depth of the user's screen to the page.

Listing 8.399 Accessing the colorDepth Property of the screen Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The color depth of this user's screen is <b>");
document.write(screen.colorDepth + '</b> bit');

// -->
</script>
```

screen.height

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

screen.height

Description

The height property of the screen object accesses the height of the user's screen in pixels.

Example

Listing 8.400 displays the height of the user's screen to the page.

Listing 8.400 Accessing the height Property of the screen Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The height of this user's screen is <b>");
document.write(screen.height + '</b> pixels');
```

Listing 8.400 Continued

```
// -->
</script>
```

Screen.pixelDepth

JavaScript1.2+
Nav4+

Syntax

`screen.pixelDepth`

Description

The `pixelDepth` property of the `screen` object accesses the number of bits per pixel of the user's screen.

Example

Listing 8.401 displays the pixel depth of the user's screen to the page.

Listing 8.401 Accessing the `pixelDepth` Property of the `screen` Object

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The pixel depth of this user's screen is <b>");
document.write(screen.pixelDepth + '</b> bit');

// -->
</script>
```

screen.unwatch()

JavaScript 1.2+
Nav4+

Syntax

`screen.unwatch(property)`

Description

The `unwatch()` method of the `screen` object is used to turn off the watch for a particular *property*.

Example

Listing 8.402 shows how the `unwatch()` method is used to stop watching the `height` property. If the `screen.height` property changes, such as in changing screen resolution, this function will be called.

Listing 8.402 Example of the `unwatch()` Method

```
<script type="text/javascript" language="JavaScript1.2">

function alertme(id, oldValue, newValue){
    document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

screen.watch("height",alertme);
screen.unwatch("height");

</script>
```

screen.watch()

JavaScript 1.2+
Nav4+

Syntax

```
screen.watch(property, function)
```

Description

The `watch()` method of the `screen` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 8.403 shows how the `watch()` method is used to start watching the `height` property. If the `screen.height` property changes, such as in changing screen resolution, this function will be called.

Listing 8.403 Example of the `watch()` method

```
<script type="text/javascript" language="JavaScript1.2">

function alertme(id, oldValue, newValue){
    document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

screen.watch("height",alertme);

</script>
```

screen.width

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

screen.width

Description

The width property of the screen object accesses the width of the user's screen in pixels.

Example

Listing 8.404 displays the width of the user's screen to the page.

Listing 8.404 *Accessing the width Property of the screen Object*

```
<script language="JavaScript1.2" type="text/javascript">
<!--

document.write("The width of this user's screen is <b>");
document.write(screen.width  + '</b> pixels');

// -->
</script>
```

Select

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Select object is one of the core JavaScript objects. Instances are created by the browser when it encounters an HTML <select> tag. In the JavaScript object hierarchy, the Select object is located at window.document.Form.Select. Table 8.30 lists the properties, methods, and event handlers used by the Select object.

Table 8.30 *Event Handlers, Methods, and Properties Used by the Select Object*

Type	Item	Description
Event Handler	onBlur	Executes code when the select box loses the focus.
	onChange	Executes code when the select box has had its value modified.

Type	Item	Description
Method	onFocus	Executes code when the select box receives the focus.
	blur()	Removes the focus from the select box.
	focus()	Gives the focus to the select box.
	handleEvent()	Invokes the handler for the event specified and was added in JavaScript 1.2.
Property	unwatch()	Used to turn off the watch for a particular property.
	watch()	Used to turn on the watch for a particular property.
	form	Returns the entire form that the select box is in.
	length	Returns the number of options in the select box.
	name	Returns the name of this select box specified by the name attribute.
	options	Returns an array containing each of the items in the select box. These items are created using the <option> HTML tag. There is also a length and selectedIndex subproperty of this property.
	selectedIndex	Returns an integer specifying the indexed location of the selected option in the select box.
	type	Returns the type of this select box specified by the type attribute. For <select> instances that contain the multiple attribute, this property returns select-multiple. Instances without this attribute return select-one. Note that this property was added in JavaScript 1.1.

Example

Listing 8.405 displays the use of the Select properties. It contains a select box and a button. When the button is clicked, a second window is opened. The values of the properties of this Select object are displayed in this window.

Listing 8.405 *Displaying the Properties of an Instance of a Select Object*

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the select box in a variable for easier access.
    var myInstance = document.myForm.mySelect;
```

Listing 8.405 Continued

```
// Open a window to store the results.
var myWin = open("", "", "width=450,height=200");

// Write the select box's properties to the window.
myWin.document.write("The length is: " + myInstance.length + "<br>");
myWin.document.write("The name is: " + myInstance.name + "<br>");
myWin.document.write("The selected option is located at position: ");
myWin.document.write(myInstance.selectedIndex + "<br>");
myWin.document.write("The type is: " + myInstance.type + "<br>");

// Note that the entire form object is passed with this property.
// This allows you to then drill down and get the value of other
// components in the form.
myWin.document.write("The form can be used to grab ");
myWin.document.write("the value of the button: ");
myWin.document.write(myInstance.form.myButton.value);

// Close the stream to the window.
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">My Favorite Sport is:
  <select name="mySelect">
    <option value="BASE">Baseball</option>
    <option value="FOOT">Football</option>
    <option value="BASKET">Basketball</option>
    <option value="SOCCER">Soccer</option>
  </select>
  <input type="button" value="Click to Process" name="myButton"
    onclick="openWin()">
</form>
</body>
</html>
```

Select.blur()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
select.blur()
```

Description

The `blur()` method of the `Select` object removes focus from the select box. This does not mean that the selected option is unselected, but rather, the focus on this option and the select box as a whole are removed. Be careful when using this method in conjunction with the `Select.focus()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.406 has a multiple select box and a button. If the user selects any of the options in the box and then clicks the Click Here to Remove Focus button, the option and entire select box will no longer have focus.

Listing 8.406 Using the `blur()` Method to Remove Focus from the Select Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the removeFocus function called by pressing the button.
function removeFocus(){

    // Remove focus from the select box.
    document.myForm.mySelect.blur();
}
// -->
</script>
</head>
<body>
<p>
<form name="myForm">
    <select name="mySelect" multiple>
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>
        <option value="BASKET">Basketball</option>
        <option value="SOCCER">Soccer</option>
    </select>
    <input type="button" value="Click Here to Remove Focus" name="myButton"
        onclick="removeFocus()">
</form>
</body>
</html>
```

Select.focus()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
select.focus()
```

Description

The `focus()` method of the `Select` object gives the focus to the select box. Be careful when using this method in conjunction with the `Select.blur()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.407 has two text boxes and two buttons. If the user clicks the first button, the cursor is set inside the first text box. If the user clicks the second text box, the cursor is set inside the second text box.

Listing 8.407 Using the `focus()` Method to Place the Cursor in the Desired Select Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the setFocus function called by pressing the button.
function setFocus(num){

    // Determine which button was clicked and set the cursor
    // in the appropriate select box.
    if(num == 1){
        document.myForm.mySelect1.focus();
    }else if(num == 2){
        document.myForm.mySelect2.focus();
    }
}
// -->
</script>
</head>
<body>
<p>
<form name="myForm">
    <select name="mySelect1" multiple>
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>
        <option value="BASKET">Basketball</option>
        <option value="SOCCER">Soccer</option>
```



```

</select>
<input type="button" value="Click to Set Cursor" name="myButton1"
       onclick="setFocus(1)">
<br>
<select name="mySelect2" multiple>
  <option value="HOCK">Hockey</option>
  <option value="RUG">Rugby</option>
  <option value="GOLF">Golf</option>
  <option value="TENNIS">Tennis</option>
</select>
<input type="button" value="Click to Set Cursor" name="myButton2"
       onclick="setFocus(2)">
</form>
</body>
</html>

```

Select.form

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.form`

Description

The `form` property of an instance of a `Select` object holds all the data of the form in which the select box is contained. This allows a developer to obtain specific information about the form in which the select box is located.

Example

Listing 8.408 has a select box and a button. When the button is clicked, three properties of the form as a whole are displayed. These properties were referenced through the `form` property of the select box.

Listing 8.408 Accessing a Form Via the `form` Property of an Instance of a `Select` Object

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

```

```

// Define the openWin function called by pressing the button.
function openWin(){

```

```

    // Place the reference to the form property of the select box
    // in a variable for easier access.
    var formData = document.myForm.mySelect.form;

```

Listing 8.408 Continued

```
// Open a window to display the results.
var myWin = open("", "", "width=450,height=200");

// Write the form properties accessed through the form
// property to the window.
myWin.document.write("The name of the form is: " + formData.name + "<br>");
myWin.document.write("The selectedIndex of the option is: ");
myWin.document.write(formData.mySelect.selectedIndex + "<br>");
myWin.document.write("The name of the button is: ");
myWin.document.write(formData.elements[1].name + "<br>");

// Close the stream to the window.
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
  <select name="mySelect" multiple>
    <option value="BASE">Baseball</option>
    <option value="FOOT">Football</option>
    <option value="BASKET">Basketball</option>
    <option value="SOCCER">Soccer</option>
  </select>
  <input type="button" value="Click to Process" name="myButton"
    onclick="openWin()">
</form>
</body>
</html>
```

Select.handleEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
select.handleEvent(event)
```

Description

The `handleEvent()` method of the `Select` object invokes the handler for the event specified. This method was added in JavaScript 1.2.

Example

Listing 8.409 has a single select box. The script tells the browser that it wants to intercept all `click` events and that it wants the `myClickHandler` function to handle them.

Within this function, the `handleEvent()` method of the select box has been specified to handle the click.

When the user clicks anywhere on the page, the `onClick` event handler in the `<select>` tag calls a function to change the selection in a select box. The change increments through the four options in the box.

Listing 8.409 Using the `handleEvent()` Method of a Select Object to Handle All Clicks on a Page

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the select box.
    window.document.myForm.mySelect.handleEvent(e);

}

// Function is called by onClick of select box.
function changeSelect(){
    if(counter > 3){
        counter = 0;
        document.myForm.mySelect.selectedIndex = counter;
    }else{
        document.myForm.mySelect.selectedIndex = counter++;
    }
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect" onclick='changeSelect() '>
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>
        <option value="BASKET">Basketball</option>
        <option value="SOCCER">Soccer</option>
```

Listing 8.409 Continued

```
</select>
</form>
</body>
</html>
```

Select.length

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
select.length
```

Description

The length property of an instance of a Select object returns the number of items in the select box.

Example

Listing 8.410 has a single select box and button. The length property of a Select object is displayed in an alert box when the button is clicked.

Listing 8.410 Using the length Property to Retrieve the Name of a Select Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Display an alert box that contains the length of the
// select box.
function getName(){
    alert("The length of this select box is " +
        document.myForm.mySelect.length);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect">
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>
        <option value="BASKET">Basketball</option>
        <option value="SOCCER">Soccer</option>
    </select>
```

```

    <input type="button" value="Get Name" name="myButton" onclick='getName()'>
  </form>
</body>
</html>

```

Select.name

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.name`

Description

The `name` property of an instance of a `Select` object returns the name of the select box. This property is often accessed via the `elements` array of a `Form` object and used to return the name of the select area. It is most useful when there are many forms on a given page, and determining the name helps you determine what function you want to perform.

Example

Listing 8.411 has a single select box and button. The `elements` array of a `Form` object is used to retrieve the name and display it in an alert box.

Listing 8.411 *Using the `name` Property to Retrieve the Name of a Select Box*

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Display an alert box that contains the name of the
// select box.
function getName(){
    alert("The name of this select box is " +
        document.myForm.elements[0].name);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect" multiple>
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>

```

Listing 8.411 Continued

```
<option value="BASKET">Basketball</option>
<option value="SOCCER">Soccer</option>
</select>
<input type="button" value="Get Name" name="myButton" onclick='getName()'>
</form>
</body>
</html>
```

Select.onBlur

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onblur="command"`

Description

The `onBlur` event handler of an instance of a `Select` object is fired when the focus is moved away from that particular select box. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onFocus` event handler or the `focus()` method.

Example

Listing 8.412 contains two select boxes, one of which has the `Blur` event intercepted within its tag and a text box. The text box is used for a counter for the number of times the `Blur` event is fired.

In the script, the event is fired and the event handler calls a function that reassigns the first select box focus. The result of this is that when a user tries to click or tab away from the first select box, the counter is incremented and the focus returns. Note that this does not mean the selected item in the select box becomes unselected, but rather the first select box gains the focus.

As you will see if you run Listing 8.412, even clicking in other windows or the URL bar increments the counter.

Listing 8.412 Example of Using the `onBlur` Event Handler

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;
```

```
// Set the focus on the first select box, and increment
// counter in the text box.
function comeBack(){
    document.myForm.mySelect1.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='comeBack() '>
<form name="myForm">
    <select name="mySelect1" multiple onBlur='comeBack() '>
        <option value="BASE">Baseball</option>
        <option value="FOOT">Football</option>
        <option value="BASKET">Basketball</option>
        <option value="SOCCER">Soccer</option>
    </select>
    <br>
    <select name="mySelect2" multiple>
        <option value="HOCK">Hockey</option>
        <option value="RUG">Rugby</option>
        <option value="GOLF">Golf</option>
        <option value="TENNIS">Tennis</option>
    </select>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>
```

Select.onChange

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

onchange="command"

Description

The onChange event handler of an instance of a Select object is fired when the option selected in the select box is changed. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other event handlers or methods that are fired when focus is placed on or away from the select box.

Example

Listing 8.413 has a single select box. If the user changes the selected option, an alert box is displayed showing the option that has been selected.

Listing 8.413 Using the `onChange` Event Handler to Display an Alert Box When an Option Is Selected in the Select Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Pop up an alert box displaying the option selected.
function changeBack(form){
    for (var i = 0; i < form.mySelect.options.length; i++) {
        if (form.mySelect.options[i].selected){
            alert("You have selected " + form.mySelect.options[i].text);
        }
    }
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect" onchange='changeBack(this.form)'\>
        <option value="HOCK">Hockey</option>
        <option value="RUG">Rugby</option>
        <option value="GOLF">Golf</option>
        <option value="TENNIS">Tennis</option>
    </select>
</form>
</body>
</html>
```

Select.onFocus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onfocus="command"`

Description

The `onFocus` event handler of an instance of a `Select` object is fired when the focus is set on that particular select box. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onBlur` event handler or the `blur()` method.

Example

Listing 8.414 contains a multiple select box and a text box. The select box has an `onFocus` event handler within its tag that is fired every time you select an option in the box. In the script, the `onFocus` event handler within the `<select>` tag calls a function

that assigns the text box focus. Each time a user clicks or tabs to the select box, the counter is incremented and the focus bar is returned to the text box.

Listing 8.414 *Example of Using the onFocus Event Handler*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the counter text box, and increment
// counter.
function sendAway(){
    document.myForm.counter.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='sendAway() '>
<form name="myForm">
    <select name="mySelect" multiple onfocus='sendAway() '>
        <option value="HOCK">Hockey</option>
        <option value="RUG">Rugby</option>
        <option value="GOLF">Golf</option>
        <option value="TENNIS">Tennis</option>
    </select>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>
```

Select.options

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.options`

Description

The `options` property of the `Select` object is an array that contains the elements of each of the options in the select box. This property is often used to retrieve properties of the options in a select box, such as the `value` or `text`.

Example

Listing 8.415 has a select box with four options in it. When an option is selected, a second window pops up. In this window, the `document.write()` method is used to write the properties of the selected option to the page.

Listing 8.415 Example of Using the options Array to Retrieve the Properties of a Selected Option in a Select Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Pop up an alert box displaying the option selected.
function infoBox(form){

    // Store the passed info into a variable for easier coding.
    var myIn = form.mySelect;

    // Open a window to store the results.
    var myWin = open("", "", "width=400,height=150");

    // Write the header info.
    myWin.document.write("The following is information about the ");
    myWin.document.write("option you selected");
    myWin.document.write('<hr height=1>');

    // Determine which option was selected.
    for (var i = 0; i < form.mySelect.options.length; i++) {
        if (form.mySelect.options[i].selected){

            // Write the option's properties to the window.
            myWin.document.write('<br><b>Value:</b> ' + myIn.options[i].value);
            myWin.document.write('<br><b>Text:</b> ' + myIn.options[i].text);
            myWin.document.close();
        }
    }
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect" onchange='infoBox(this.form) '>
        <option value="HOCK">Hockey</option>
        <option value="RUG">Rugby</option>
        <option value="GOLF">Golf</option>
        <option value="TENNIS">Tennis</option>
    </select>
```

```
</form>
</body>
</html>
```

Select.options.length

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.options.length`

Description

The `length` property of the `options` array of the `Select` object returns the number of options in that instance of a select box.

Example

Listing 8.416 has a single select box. When the user makes a selection, the `onChange` event handler is used to pop up an alert box that displays the number of options in this select box.

Listing 8.416 *Accessing the `length` Property of the `options` Array of a `Select` Object*

```
<form name="myForm">
  <select name="mySelect" onchange='alert(mySelect.options.length) '>
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
    <option value="TENNIS">Tennis</option>
  </select>
</form>
```

Select.options.selectedIndex

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.options.selectedIndex`

Description

The `selectedIndex` property of the `options` array of the `Select` object returns the index number of the selected option in that instance of a select box.

Example

Listing 8.417 has a single select box. When the user makes a selection, the `onChange` event handler is used to pop up an alert box that displays the index number of the selected option in this select box.

Listing 8.417 Accessing the *selectedIndex* *Property of the* options *Array of a* Select *Object*

```
<form name="myForm">
  <select name="mySelect" onchange='alert(mySelect.options.selectedIndex) '>
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
    <option value="TENNIS">Tennis</option>
  </select>
</form>
```

Select.options.value

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.options.value`

Description

The `value` property of the options array of the `Select` object returns the value of the option that is selected in that instance of a select box.

Example

Listing 8.418 has a single select box. When the user makes a selection, the `onChange` event handler is used to pop up an alert box that displays the value of the selected option in this select box.

NOTE

Different browsers might fire this event at different times, so it is possible that the alert box is displayed differently.

Listing 8.418 Accessing the *value* *Property of the* options *Array of a* Select *Object*

```
<form name="myForm">
  <select name="mySelect"
    onchange='alert(mySelect.options[selectedIndex].value) '>
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
```

```

    <option value="TENNIS">Tennis</option>
  </select>
</form>

```

Select.selectedIndex

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`select.selectedIndex`

Description

The `selectedIndex` property of the `Select` object returns the index number of the selected option in that instance of a select box. If this property is used to access a multiple select box, it will return the index number of the first selected item.

Example

Listing 8.419 has a single select box. When the user makes a selection, the `onChange` event handler is used to pop up an alert box that displays the index number of the selected option in this select box.

Listing 8.419 *Accessing the `selectedIndex` Property of the `Select` Object*

```

<form name="myForm">
  <select name="mySelect" onchange='alert(mySelect.selectedIndex) '>
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
    <option value="TENNIS">Tennis</option>
  </select>
</form>

```

Select.type

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

`select.type`

Description

The `type` property of an instance of a `Select` object returns the type of the select box. This is either `select-multiple`, if the `multiple` attribute is set in the `<select>` tag, or `select-one` if it is not.

Example

Listing 8.420 has a select box and a button. When the button is clicked, an alert box is popped up that displays the type property of the select box.

Listing 8.420 Displaying the type Property in an Alert Box

```
<html>
<head>
<script language="JavaScript1.1" type="text/javascript">
<!--

// Display an alert box that contains the type of the
// select box.
function getType(){
    alert("The name of this text box is " +
        document.myForm.elements[0].type);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <select name="mySelect">
        <option value="HOCK">Hockey</option>
        <option value="RUG">Rugby</option>
        <option value="GOLF">Golf</option>
        <option value="TENNIS">Tennis</option>
    </select>
    <input type="button" value="Get Type" name="myButton" onclick='getType()'>
</form>
</body>
</html>
```

Select.unwatch()

JavaScript 1.2+
Nav4+

Syntax

```
select.unwatch(property)
```

Description

The `unwatch()` method of the `Select` object is used to turn off the watch for a particular *property*.

Example

Listing 8.421 shows how the `unwatch()` method is used to stop watching the value property.

Listing 8.421 Example of the `unwatch()` Method

```

<html>
<head>
  <script type="text/javascript" language="JavaScript1.2">

    function alertme(id, oldValue, newValue){
      document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
      document.writeln(newValue + "<br>");
      return newValue;
    }

    function check(){
      document.myForm.mySelect.watch("value", alertme);
      document.myForm.mySelect.unwatch("value");
      document.myForm.mySelect.value="GOLF";
    }
  </script>
</head>
<body onload="check()">
<form name="myForm">
  <select name="mySelect">
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
    <option value="TENNIS">Tennis</option>
  </select>
</form>
</body>
</html>

```

Select.watch()

JavaScript 1.2+
Nav4+

Syntax

`select.watch(property, function)`

Description

The `watch()` method of the `Select` object is used to turn on the watch for a particular *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 8.422 shows how the `watch()` method is used to start watching the *value* property.

Listing 8.422 Example of the watch() Method

```
<html>
<head>
  <script type="text/javascript" language="JavaScript1.2">

    function alertme(id, oldValue, newValue){
      document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
      document.writeln(newValue + "<br>");
      return newValue;
    }

    function check(){
      document.myForm.mySelect.watch("value", alertme);
      document.myForm.mySelect.value="GOLF";
    }
  </script>
</head>
<body onload="check()">
<form name="myForm">
  <select name="mySelect">
    <option value="HOCK">Hockey</option>
    <option value="RUG">Rugby</option>
    <option value="GOLF">Golf</option>
    <option value="TENNIS">Tennis</option>
  </select>
</form>
</body>
</html>
```

Style

JavaScript1.2,
Nav4

Syntax

Core client-side JavaScript object.

```
document.classes.className.tagName
document.ids.idName
document.tags.tagName
document.contextual(tag1, tag2)
```

Description

The Style object is a core client-side object within the Navigator 4 browser. Instances of this object exist and are accessible through four different methods. Three of these methods, `document.ids`, `document.classes`, and `document.tags`, represent associative arrays that contains the ids, classes, or tags, respectively, associated with the various style access options. The `document.contextual()` method, on the other hand,

finds instances of *tag1* that occur within instances of *tag2*, and then creates an array filled with matching instances.

Table 8.31 contains a list of properties and methods for this object.

Table 8.31 *Properties and Methods of the Style Object*

Type	Item	Description
Method	<code>borderWidths()</code>	Sets border width of an HTML element.
	<code>margins()</code>	Sets minimal distance between the sides of an HTML element and the sides of adjacent elements.
	<code>padding()</code>	Sets space to insert between the sides of an element and its content, such as text or an image.
	<code>unwatch()</code>	Turns off the watch for a particular property.
Property	<code>watch()</code>	Turns on the watch for a particular property.
	<code>align</code>	Sets alignment of an HTML element within its parent.
	<code>backgroundColor</code>	Sets solid background color for an element.
	<code>backgroundImage</code>	Sets background image for an HTML element.
	<code>borderBottomWidth</code>	Sets width of the bottom border of an HTML element.
	<code>borderColor</code>	Sets color of the border of an HTML element.
	<code>borderLeftWidth</code>	Sets width of the left border of an HTML element.
	<code>borderRightWidth</code>	Sets width of the right border of an HTML element.
	<code>borderStyle</code>	Sets style of border around a block-level HTML element.
	<code>borderTopWidth</code>	Sets width of the top border of an HTML element.
	<code>clear</code>	Sets sides of an HTML element that allow floating elements.
	<code>color</code>	Sets color of the text in an HTML element.
	<code>display</code>	Overrides the default display of an element and specifies whether it will appear in line, as a block-level element, or as a block-level list item.
	<code>fontFamily</code>	Sets font family, such as Helvetica or Arial, for an HTML text element.
	<code>fontSize</code>	Sets font size for an HTML text element.
	<code>fontStyle</code>	Sets style of the font of an HTML element.
	<code>fontWeight</code>	Sets weight of the font of an HTML element.
	<code>lineHeight</code>	Sets distance between the baselines of two adjacent lines of block-level type.
	<code>listStyleType</code>	Sets style of bullet displayed for list items.

Table 8.31 Continued

Type	Item	Description
	marginBottom	Sets minimum distance between the bottom of an HTML element and the top of an adjacent element.
	marginLeft	Sets minimum distance between the left side of an HTML element and the right side of an adjacent element.
	marginRight	Sets minimum distance between the right side of an HTML element and the left side of an adjacent element.
	marginTop	Sets minimum distance between the top of an HTML element and the bottom of an adjacent element.
	paddingBottom	Sets space to insert between the bottom of an element and its content, such as text or an image.
	paddingLeft	Sets space to insert between the left side of an element and its content, such as text or an image.
	paddingRight	Sets space to insert between the right side of an element and its content, such as text or an image.
	paddingTop	Sets space to insert between the top of an element and its content, such as text or an image.
	textAlign	Sets alignment of an HTML block-level text element.
	textDecoration	Sets special effects, such as blinking, strike-outs, and underlines, added to an HTML text element.
	textIndent	Sets length of indentation appearing before the first formatted line of a block-level HTML text element.
	textTransform	Sets case of an HTML text element.
	whiteSpace	Sets whether whitespace within an HTML element should be collapsed.
	width	Sets width of a block-level HTML element.

NOTE

Navigator 4 is the only browser that supports this object, which was deprecated in Netscape 6. You should only use it if you are scripting HTML or XHTML documents that require backward compatibility with Navigator 4.

Example

Listing 8.423 demonstrates three different ways to use `document.ids` to control the style result.

Listing 8.423 Using the Style Object

```
<html>

<style type="text/css">
  #TEXTFORMAT {font-style: italic;}
</style>

<p id="TEXTFORMAT">After first STYLE tag.<p>

<style type="text/javascript">
  ids.TEXTFORMAT.fontWeight = "bold";
</style>

<p id="TEXTFORMAT">After second STYLE tag.<p>

<script type="text/javascript" language="JavaScript">
<!--
document.ids.TEXTFORMAT.textDecoration = "underline";
document.classes.no-under.a.textDecoration = "none";
// -->
</script>

<p id="TEXTFORMAT">After SCRIPT tag.<p>
<a href="http://www.purejavascript.com" class="no-under">No underline</a>

</html>
```

Style.align

JavaScript1.2
Nav4

Syntax

`style.name.align`

Description

The `align` property specifies the alignment of an element within its parent as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The `align` property can be assigned one of the following values: `left`, `right`, or `none`.

 NOTE

The `align` property is referred to as `float` when using CSS syntax. JavaScript could not use the word “float” because it is a reserved word.

Example

Listing 8.424 uses the `align` property to align a paragraph to the right of its parent, the document.

Listing 8.424 *Aligning Right with the align Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that aligns right.
document.ids.RIGHT.align = "right";

// -->
</script>

<p id="RIGHT">Send me to the right!</p>

<p>Where am I?</p>

</html>
```

Style.backgroundColor

JavaScript1.2
Nav4

Syntax

`style.name.backgroundColor`

Description

The `backgroundColor` property specifies the background color as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.425 uses the `backgroundColor` property to make the background color around an anchor red.

Listing 8.425 *Setting the Background Color with the* backgroundColor *Property*

```

<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that contains the background color
document.ids.BG.backgroundColor = "red";

// -->
</script>

<a id="BG" name="THIS">This</a> is an anchor that uses
styles from the BG id.

</html>

```

Style.backgroundColor

JavaScript1.2

Nav4

Syntax`style.name.backgroundColor`**Description**

The `backgroundColor` property specifies the background image of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

The code in Listing 8.426 makes the image `logo.gif` the background for the header text.

Listing 8.426 *Setting the Background Image with the* backgroundImage *Property*

```

<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that uses an image as a background
document.ids.BGI.backgroundColor = "logo.gif";

```

Listing 8.426 Continued

```
// -->
</script>

<h2 id="BGI">Look at my background image!</h2>

</html>
```

Style.borderBottomWidth

JavaScript1.2
Nav4

Syntax

`style.name.borderBottomWidth`

Description

The `borderBottomWidth` property specifies the width of the bottom border of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.427 uses the `borderBottomWidth` property to set the size of bottom border that surrounds a text header. Notice that only the bottom portion of the border is shown.

**Listing 8.427 Setting the Bottom Border Width with the
`borderBottomWidth` Property**

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a bottom border
document.ids.BBW.borderBottomWidth = "10";

// -->
</script>

<h2 id="BBW">This text has a bottom border</h2>

</html>
```

Style.borderColor

JavaScript1.2

Nav4

Syntax

style.name.borderColor

Description

The `borderColor` property specifies the color of the border of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.428 uses the `borderColor` property to set the color of the border that surrounds a text header to blue.

Listing 8.428 *Setting the Border Color with the* `borderColor` *Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a left border
document.ids.a.borderWidths(10);
document.ids.a.borderColor = "blue";

// -->
</script>

<h2 id="a">This text has a blue border</h2>

</html>
```

Style.borderLeftWidth

JavaScript1.2

Nav4

Syntax

style.name.borderLeftWidth

Description

The `borderLeftWidth` property specifies the width of the left border of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.429 uses the `borderLeftWidth` property to set the size of the left border that surrounds a text header. Notice that only the left portion of the border is shown.

Listing 8.429 *Setting the Left Border Width with the `borderLeftWidth` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a left border
document.ids.BLW.borderLeftWidth = "10";

// -->
</script>

<h2 id="BLW">This text has a left border</h2>

</html>
```

Style.borderRightWidth

JavaScript1.2
Nav4

Syntax

`style.name.borderRightWidth`

Description

The `borderRightWidth` property specifies the width of the right border of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.430 uses the `borderRightWidth` property to set the size of right border that surrounds a text header. Notice that only the right portion of the border is shown.

Listing 8.430 *Setting the Right Border Width with the `borderRightWidth` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--
```



```
//Create a style sheet id that creates a right border.
document.ids.BRW.borderRightWidth = "10";

// -->
</script>

<h2 id="BRW">This text has a right border</h2>

</html>
```

Style.borderStyle

JavaScript1.2

Nav4

Syntax

`style.name.borderStyle`

Description

The `borderStyle` property specifies the style of the border that surrounds an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The `borderStyle` property can be assigned one of the following values: `none`, `solid`, `double`, `inset`, `outset`, `groove`, and `ridge`.

Example

Listing 8.431 uses the `borderStyle` property to set the border around a text header to a double line.

Listing 8.431 *Setting the Border Style with the `borderStyle` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a double border.
document.ids.a.borderWidths(10);
document.ids.a.borderColor = "red";
document.ids.a.borderStyle = "double";

// -->
</script>

<h2 id="a">This text has a double border</h2>

</html>
```

Style.borderTopWidth

JavaScript1.2

Nav4

Syntax

`style.name.borderTopWidth`

Description

The `borderTopWidth` property specifies the width of the top border of an element as associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.432 uses the `borderTopWidth` property to set the size of top border that surrounds a text header. Notice that only the top portion of the border is shown.

Listing 8.432 *Setting the Top Border Width with the* `borderTopWidth` *Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a top border.
document.ids.BTW.borderTopWidth = "10";

// -->
</script>

<h2 id="BTW">This text has a top border</h2>

</html>
```

Style.borderWidths()

JavaScript1.2

Nav4

Syntax

```
style.borderWidths(top, right, bottom, left)
style.borderWidths(top-bottom, right-left)
style.borderWidths(all)
```

Description

The `borderWidths()` method specifies the width of the border that surrounds an element as associated with *style*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Setting border widths is equivalent to setting the `borderBottomWidth`, `borderLeftWidth`, `borderRightWidth`, and `borderTopWidth` properties. Table 8.32 lists all the arguments associated with the `borderWidths()` method.

Table 8.32 Arguments Associated with the `borderWidths()` Method

Item	Description
<code>top</code>	Width of the top border
<code>right</code>	Width of the right border
<code>left</code>	Width of the left border
<code>bottom</code>	Width of the bottom border
<code>top-bottom</code>	Width of both the top and bottom border
<code>left-right</code>	Width of both the left and right border
<code>all</code>	Width of all four sides of the border

Example

Listing 8.433 uses the `borderWidths()` method to set the width of the border that surrounds a text header.

Listing 8.433 Setting All the Border Widths with the `borderWidths()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that creates a border.
document.ids.a.borderWidths(5,10,15,20);

// -->
</script>

<h2 id="a">This text has a border</h2>

</html>
```

Style.clear

JavaScript1.2

Nav4

Syntax

`style.clear`

Description

The `clear` property specifies the sides of an element where floating elements (elements that define the `align` property) are not allowed. Using dot notation, this property is associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The `clear` property can be assigned one of the following values: `left`, `right`, `both`, and `none`.

Example

Listing 8.434 uses the `clear` property to prevent floating elements from being on either side.

Listing 8.434 *Using the `clear` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
  <!--

  //Create style sheet classes and id that set alignments.
  document.classes.LEFT.all.align = "left";
  document.classes.LEFT.all.backgroundColor = "yellow";
  document.classes.RIGHT.all.align = "right";
  document.classes.RIGHT.all.backgroundColor = "red";
  document.ids.KEEPAWAY.clear = "both";

  // -->
</script>

<p class="LEFT">Send me to the left!</p>
<p class="RIGHT">Send me to the right!</p>
<p>I like other elements next to me!</p>

<p class="LEFT">Send me to the left!</p>
<p class="RIGHT">Send me to the right!</p>
<p id="KEEPAWAY">I don't like other elements next to me!</p>

</html>
```

Style.color

JavaScript1.2

Nav4

Syntax

`style.color`

Description

The `color` property specifies the foreground color of an element as associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.435 uses the `color` property to set the color of various text.

Listing 8.435 Set the Color of Text Using the `color` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create style sheet classes that define colors.
document.ids.BLUE.color = "blue";
document.ids.RED.color = "red";

// -->
</script>

<p id="BLUE">The blue boat floated on the blue ocean.</p>
<p id="RED">The red car stopped at a red stop sign.</p>

</html>
```

Style.display

JavaScript1.2

Nav4

Syntax

`style.display`

Description

The `display` property specifies an element to be displayed when associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The display property can be assigned one of the following values: inline, block, list-item, and none.

Specifying an inline value is equivalent to using the tag. The block value is the same as creating header text with the <h1>–<h6> tags. Using list-item is equivalent to using the tag. If none is specified, the element is not displayed.

Example

Listing 8.436 uses the display property to create header text. block consists of a line break and resetting margins to their default values. Notice that in this example a carriage return is automatically entered before the bold tag.

Listing 8.436 *Set the display Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines a block display.
document.ids.H.display = "block";

// -->
</script>

<u id="H">A BIG HEADER</u>

Just some regular text following the Big Header.

</html>
```

Style.fontFamily

JavaScript1.2
Nav4

Syntax

style.fontFamily

Description

The fontFamily property specifies the font an element should use when associated with style. Because style can be created using the document.classes, document.ids, document.tags, or document.contextual() methods, you should check the entries for these items for more information on the exact syntax. More than one font can be specified, in case a particular font has not been loaded.

Example

Listing 8.437 uses the fontFamily property to create text with an Arial font. If that font is not available, Helvetica is used.

Listing 8.437 Set the `fontFamily` Property

```

<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines a font.
document.ids.F.fontFamily = "Arial, Helvetica";

// -->
</script>

<p id="F">Do you like this font?</p>

</html>

```

Style.fontSize

JavaScript1.2
Nav4

Syntax

`style.fontSize`

Description

The `fontSize` property specifies the size of fonts used by an element when associated with *name*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax. The values that can be used to set font size can be divided into four categories, as shown in Table 8.33.

Table 8.33 Font Sizes

Category	Value
Absolute	xx-small
	x-small
	small
	medium
	large
	x-large
	xx-large
Relative	smaller (relative to font size of parent)
	larger (relative to font size of parent)
Length	A number followed by a unit of measurement
Percentage	The size relative to font size of parent

Example

Listing 8.438 uses absolute font sizing to define the `fontSize` property and associate with text.

Listing 8.438 *Set the `fontSize` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines a font size.
document.ids.XXS.fontSize = "xx-small";
document.ids.XXL.fontSize = "xx-large";

// -->
</script>

<p id="XXS">Extra, extra small text</p>
<p id="XXL">Extra, extra large text</p>

</html>
```

Style.fontStyle

JavaScript1.2
Nav4

Syntax

`style.fontSize`

Description

The `fontSize` property specifies the font style used by an element when associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The `fontSize` property can be assigned one of the following values: `normal` or `italic`.

Example

Listing 8.439 uses the `fontSize` property to italicize the header text.

Listing 8.439 *Set the `fontSize` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
```



```
<!--

//Create a style sheet id that defines a font style.
document.ids.I.fontStyle = "italic";

// -->
</script>

<h2 id="I">Italicized Heading</h2>

</html>
```

Style.fontWeight

JavaScript1.2
Nav4

Syntax

`style.fontWeight`

Description

The `fontWeight` property specifies the font weight used by an element when associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

The `fontStyle` property can be assigned one of the following values: `normal`, `bold`, `bolder`, `lighter`, or a number from 100 to 900.

Example

Listing 8.440 uses the `fontWeight` property to set different levels of weight on various text.

Listing 8.440 *Set the `fontWeight` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines a font style.
document.ids.W1.fontWeight = 100;
document.ids.W4.fontWeight = 400;
document.ids.W9.fontWeight = 900;

// -->
</script>
```

Listing 8.440 Continued

```
<p id="W1">Font weight of 100</p>
<p id="W4">Font weight of 400</p>
<p id="W9">Font weight of 900</p>

</html>
```

Style.lineHeight

JavaScript1.2
Nav4

Syntax

`style.lineHeight`

Description

The `lineHeight` property specifies the distance between two lines that are next to each other. Using dot notation, the property is associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Four types of values are valid for the `lineHeight` property: number, length, percentage, and the value `normal`.

When a number is given without a unit of measure, it is multiplied by the font size of the element to give the line height, while including a unit of measure after the number specifies length. A percentage is designated by including a percent sign (%) after the number to represent the line height as it relates to its parent.

Example

Listing 8.441 uses the `lineHeight` property to set the distance between lines. In the "D1" instance, it is set to 1 inch. In the "D2" instance, it is set to 50 point.

Listing 8.441 Set the `lineHeight` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines a line height.
document.ids.D1.lineHeight = "1in";
document.ids.D2.lineHeight = "50pt";

// -->
</script>
```

```

<p>This is the first line of text</p>
<p id="D1">This is a second line of text.</p>
<p>This is a third line of text.</p>
<p id="D2">This is a fourth line of text.</p>
<p>This is a fifth line of text.</p>

</html>

```

Style.listStyleType

JavaScript1.2
Nav4

Syntax

`style.listStyleType`

Description

The `listStyleType` property specifies the format of list items elements that are associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Nine types of values are valid for the `listStyleType` property: `disc`, `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, and `none`.

NOTE

The `listStyleType` property is only valid if the element also has the `display` property set to `list-item`.

Example

Listing 8.442 uses the `listStyleType` property to format a list of items.

Listing 8.442 Set the `listStyleType` Property

```

<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet class and id that define a list format.
document.classes.LIST.all.display = "list-item";
document.ids.FORMAT.listStyleType = "upper-roman";

// -->
</script>

```

Listing 8.442 Continued

```
<h2>Presentation Outline</h2>

<ol class="LIST" id="FORMAT">
  <li>Introduction</li>
  <li>Overview of new product</li>
  <li>Cost of product</li>
  <li>Conclusion</li>
</ol>

</html>
```

Style.marginBottom

JavaScript1.2
Nav4

Syntax

`style.marginBottom`

Description

The `marginBottom` property specifies the minimum distance between the bottom border of an element and the top border of another element. Dot notation is used to associate this property with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.443 uses the `marginBottom` property to set the distance between adjacent elements.

Listing 8.443 Set the Bottom Margin with the `marginBottom` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet class and id that define the bottom margin.
document.classes.FORMAT.all.borderWidths(10);
document.ids.margin.marginBottom = 40;

// -->
</script>

<b>My margins are set automatically</b>
```

```
<b class="FORMAT" id="margin">I have a large bottom margin!</b>
<b>My margins are set automatically</b>

</html>
```

Style.marginLeft

JavaScript1.2
Nav4

Syntax

`style.marginLeft`

Description

The `marginLeft` property specifies the distance between the left border of an element and the right border of another element. Dot notation is used to associate this property with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.444 uses the `marginLeft` property to set the distance between adjacent elements. Notice the distance between the border and the right edge of the browser window.

Listing 8.444 *Set the Left Margin with the `marginLeft` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the left margin.
document.classes.FORMAT1.all.borderWidths(10);
document.ids.margin.marginLeft = 40;
document.classes.FORMAT2.all.borderWidths(10);

// -->
</script>

<p class="FORMAT1" id="margin">I have a large left margin!</p>
<p class="FORMAT2">I have an automatic left margin</p>

</html>
```

Style.marginRight

JavaScript1.2

Nav4

Syntax

`style.marginRight`

Description

The `marginRight` property specifies the distance between the right border of an element and the left border of another element. Dot notation is used to associate this property with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.445 uses the `marginRight` property to set the distance between adjacent elements. Notice the distance between the border and the right edge of the browser window.

Listing 8.445 Set the Right Margin with the `marginRight` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the right margin.
document.classes.FORMAT.all.borderWidths(10);
document.ids.margin1.marginRight = 40;
document.ids.margin2.marginRight = 0;

// -->
</script>

<p class="FORMAT" id="margin1">I have a large right margin!</p>
<p class="FORMAT" id="margin2">I have no right margin</p>

</html>
```

Style.margins()

JavaScript1.2

Nav4

Syntax

```
style.margins(top, right, bottom, left)
style.margins(top-bottom, right-left)
style.margins(all)
```

Description

The `margins()` method specifies the margin distance between the border of an element and the border of adjacent elements. Dot notation is used to associate the property with *style*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Setting margin widths is equivalent to setting the `marginBottom`, `marginLeft`, `marginRight`, and `marginTop` properties. Table 8.34 lists all the arguments associated with the `margins()` method.

Table 8.34 Arguments Associated with the `margins()` Method

Item	Description
<code>top</code>	Width of the top margin
<code>right</code>	Width of the right margin
<code>left</code>	Width of the left margin
<code>bottom</code>	Width of the bottom margin
<code>top-bottom</code>	Width of both the top and bottom margins
<code>left-right</code>	Width of both the left and right margins
<code>all</code>	Width of all four margins

Example

Listing 8.446 uses the `margins()` method to set the margin widths around a text element.

Listing 8.446 All Margins Set with the `margins()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the margins.
document.classes.FORMAT.all.borderWidths(10);
document.ids.margin.margins(50);

// -->
</script>

<h2 class="FORMAT" id="margin">This text has margins set to 50</h2>

</html>
```

Style.marginTop

JavaScript1.2

Nav4

Syntax

`style.marginTop`

Description

The `marginTop` property specifies the minimum distance between the top border of an element and the bottom border of another element. Dot notation is used to associate this property with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.447 uses the `marginTop` property to set the minimum distance between adjacent elements.

Listing 8.447 Set the Top Margin with the `marginTop` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the top margin.
document.classes.FORMAT.all.borderWidths(10);
document.ids.margin.marginTop = 40;

// -->
</script>

<b>My margins are set automatically</b>
<b class="FORMAT" id="margin">I have a large top margin!</b>
<b>My margins are set automatically</b>

</html>
```

Style.paddingBottom

JavaScript1.2

Nav4

Syntax

`style.paddingBottom`

Description

The `paddingBottom` property specifies the minimum distance between the bottom border of an element and its content. Using dot notation, the property is associated with *style*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.448 uses the `paddingBottom` property to set the distance between a line of text and its border.

Listing 8.448 Set the Bottom Padding with the `paddingBottom` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the bottom padding.
document.classes.FORMAT.all.borderWidths(10)
document.ids.PAD.paddingBottom = "40";

// -->
</script>

<p class="FORMAT" id="PAD">This text has extra padding on the bottom.</p>

</html>
```

Style.paddingLeft

JavaScript1.2
Nav4

Syntax

style.paddingLeft

Description

The `paddingLeft` property specifies the minimum distance between the left border of an element and its content. Using dot notation, the property is associated with *style*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.449 uses the `paddingLeft` property to set the distance between a line of text and its border.

Listing 8.449 Set the Left Padding with the `paddingLeft` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the left padding.
document.classes.FORMAT.all.borderWidths(10)
document.ids.PAD.paddingLeft = "40";

// -->
</script>

<p class="FORMAT" id="PAD">This text has extra padding on the left.</p>

</html>
```

Style.paddingRight

JavaScript1.2
Nav4

Syntax

`style.paddingRight`

Description

The `paddingRight` property specifies the distance between the right border of an element and its content. Using dot notation, the property is associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.450 uses the `paddingRight` property to set the distance between a line of text and its border.

Listing 8.450 Set the Right Padding with the `paddingRight` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the right padding.
document.classes.FORMAT.all.borderWidths(10)
document.ids.PAD.paddingRight = "40";
```

```
// -->
</script>

<p class="FORMAT" id="PAD">This text has extra padding on the right.</p>

</html>
```

Style.paddings()

JavaScript1.2
Nav4

Syntax

```
style.paddings(top, right, bottom, left)
style.paddings(top-bottom, right-left)
style.paddings(all)
```

Description

The `paddings()` method specifies the distance between the borders of an element and its content. Using dot notation, the property is associated with *style*. Because *style* can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax. Setting the sizes is equivalent to setting the `paddingBottom`, `paddingLeft`, `paddingRight`, and `paddingTop` properties. Table 8.35 lists all the arguments associated with this method.

Table 8.35 Arguments Associated with the `paddings()` Method

Item	Description
<code>top</code>	Padding between top border and content
<code>right</code>	Padding between right border and content
<code>left</code>	Padding between left border and content
<code>bottom</code>	Padding between bottom border and content
<code>top-bottom</code>	Padding between the content and both the top and bottom borders
<code>left-right</code>	Padding between the content and both the left and right borders
<code>all</code>	Padding between content and all four sides of the border

Example

Listing 8.451 uses the `paddings()` method to set the distance between text and all sides of its border.

Listing 8.451 Set All Paddings with the `padding()` Method

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines all paddings.
document.classes.FORMAT.all.borderWidths(5);
document.ids.PAD.padding(0,20,40,60);

// -->
</script>

<p class="FORMAT" id="PAD">This text has padding on all sides of its
border.</p>

</html>
```

Style.paddingTop

JavaScript1.2

Nav4

Syntax

`style.paddingTop`

Description

The `paddingTop` property specifies the distance between the left border of an element and its content. Using dot notation, the property is associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Example

Listing 8.452 uses the `paddingTop` property to set the distance between a line of text and its border.

Listing 8.452 Set Top Padding with the `paddingTop` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the top padding.
document.classes.FORMAT.all.borderWidths(10)
document.ids.PAD.paddingTop = "40";
```

```
// -->
</script>

<p class="FORMAT" id="PAD">This text has extra padding on the top.</p>

</html>
```

Style.textAlign

JavaScript1.2
Nav4

Syntax

`style.textAlign`

Description

The `textAlign` property specifies the alignment of text within an element as associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Four types of values are valid for the `textAlign` property: `left`, `right`, `center`, and `justify`.

Example

Listing 8.453 uses the `textAlign` property to align the text to the right.

Listing 8.453 Set the `textAlign` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the right alignment.
document.classes.FORMAT.all.borderWidths(10)
document.ids.RIGHT.textAlign = "right";

// -->
</script>

<p class="FORMAT" id="RIGHT">This text is aligned right.</p>

</html>
```

Style.textDecoration

JavaScript1.2

Nav4

Syntax

`style.textDecoration`

Description

The `textDecoration` property specifies the type of decoration that is added to text as associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Four types of values are valid for the `textDecoration` property: `underline`, `line-through`, `blink`, and `none`.

Example

Listing 8.454 uses the `textDecoration` property to put a line through a line of text.

Listing 8.454 *Set the `textDecoration` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the text decoration.
document.ids.CROSSOUT.textDecoration = "line-through";

// -->
</script>

<p id="CROSSOUT">This text is crossed out.</p>

</html>
```

Style.textIndent

JavaScript1.2

Nav4

Syntax

`style.textIndent`

Description

The `textIndent` property specifies that the indentation should appear before text as associated with `style`. Because `style` can be created using the `document.classes`,

`document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax. The property is assigned a number that represents either length or a percentage.

Example

Listing 8.455 uses the `textIndent` property to set the text indentation to 1 inch.

Listing 8.455 *Set the `textIndent` Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines the text indentation.
document.ids.INDENT.textIndent = "1in";

// -->
</script>

<p>This text has no indention.</p>
<p id="INDENT">This text is indented 1 inch.</p>

</html>
```

Style.textTransform

JavaScript1.2
Nav4

Syntax

`style.textTransform`

Description

The `textTransform` property specifies the transformation that should be applied to text as associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Four types of values are valid for the `textTransform` property: `capitalize`, `uppercase`, `lowercase`, and `none`.

Example

Listing 8.456 uses the `textTransform` property to capitalize the first letter of every word in the sentence.

Listing 8.456 Set the `textTransform` Property

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines capitalization.
document.ids.CAP.textTransform = "capitalize";

// -->
</script>

<p id="CAP">This line demonstrates the ability to capitalize words.</p>

</html>
```

Style.unwatch()

JavaScript 1.2
Nav4

Syntax

```
style.unwatch(property)
```

Description

The `unwatch()` method of the `Style` object is used to turn off the watch for a particular *property*.

Example

Listing 8.457 shows how the `unwatch()` method is used to stop watching the user-defined property *textTransform*.

Listing 8.457 Example of the `unwatch()` method

```
<html>
<head>
<script type="text/javascript" language="JavaScript1.2">

function alertme(id, oldValue, newValue){
    document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

document.ids.CAP.watch("textTransform", alertme);
document.ids.CAP.textTransform = "capitalize";
document.ids.CAP.unwatch("textTransform");
document.ids.CAP.textTransform = "lowercase";
```



```
// -->
</script>
</head>
<body>
<p id="CAP">This line demonstrates the ability to capitalize words.</p>
</body>
</html>
```

Style.watch()

JavaScript1.2
Nav4

Syntax

```
style.watch(property, function)
```

Description

The `watch()` method of the `Style` object is used to turn on the watch for a particular *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 8.458 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 8.458 Example of the `watch()` Method

```
<html>
<head>
<script type="text/javascript" language="JavaScript1.2">

function alertme(id, oldValue, newValue){
    document.writeln("ID ( " + id + " ) changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
    return newValue;
}

document.ids.CAP.watch("textTransform", alertme);
document.ids.CAP.textTransform = "capitalize";

// -->
</script>
</head>
<body>
<p id="CAP">This line demonstrates the ability to capitalize words.</p>
</body>
</html>
```

Style.whiteSpace

JavaScript1.2

Nav4

Syntax

`style.whiteSpace`

Description

The `whiteSpace` property specifies how whitespace should be handled within an element. Using dot notation, the property can be associated with `style`. Because `style` can be created using the `document.classes`, `document.ids`, `document.tags`, or `document.contextual()` methods, you should check the entries for these items for more information on the exact syntax.

Two types of values are valid for the `whiteSpace` property: `normal` and `pre`.

Example

Listing 8.459 uses the `whiteSpace` property to make whitespace collapsed within the text element.

Listing 8.459 *Set the whiteSpace Property*

```
<html>

<script type="text/javascript" language="JavaScript">
<!--

//Create a style sheet id that defines whitespace.
document.ids.NOPRE.whiteSpace = "normal";

// -->
</script>

<p><pre>This line used the PRE tag to format whitespace.</pre></p>
<p id="NOPRE">This line should have collapsed whitespace.</p>

</html>
```

Style.width

JavaScript1.2

Nav4

Syntax

`style.width`

Description

The width property of the Style object is used to specify the width of a block-level HTML element. The property itself can take a numerical length, a percentage, or the keyword auto, which tells Navigator 4 to determine the width automatically. The following shows a brief example of syntax:

```
// length
myStyle.width = 12pt;

// percentage
myStyle.width = 75%;

// automatically format
myStyle.width = auto;
```

Example

Listing 8.460 shows how we can use the width property to set the border of a <p> element to 12 point.

Listing 8.460 *Using the width Property to Set the Border Width of an Element*

```
<html>
<head>
  <script type="text/javascript" language="JavaScript">
    <!--

      //Set the border of our <p> tag to 12 point.
      document.ids.border.width = 12pt;

    // -->
  </script>
</head>
<body>
  <p id="border">This has a 12 point border.</p>
</body>
</html>
```

Submit

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Submit object is one of the core JavaScript objects. Instances are created by the browser when it encounters an HTML <input> tag with the type attribute set to

submit. In the JavaScript object hierarchy, the Submit object is located at window.document.Form.Submit. Table 8.36 lists the properties, methods, and events used by the Submit object.

NOTE

Starting with Navigator 4, if you submit a form to a mailto: or news: protocol, the UniversalSendMail security privilege must be set.

Table 8.36 *Event Handlers, Methods, and Properties Used by the Submit Object*

Type	Item	Description
Event Handler	onBlur	Executes code when the submit button loses focus. This event handler was added in JavaScript 1.1.
	onClick	Executes code when the submit button is clicked.
	onFocus	Executes code when the submit button receives the focus. This event handler was added in JavaScript 1.1.
Method	blur()	Removes focus from the submit button. This method was added in JavaScript 1.1.
	click()	Simulates a mouse click on the submit button.
	focus()	Gives the focus to the submit button. This method was added in JavaScript 1.1.
	handleEvent()	Invokes the handler for the event specified and was added in JavaScript 1.2.
	unwatch()	Used to turn off the watch for a particular property.
Property	watch()	Used to turn on the watch for a particular property.
	form	Returns the entire form that the submit button is in.
	name	Returns the name of the submit button specified by the name attribute.
	type	Returns the type of the submit button specified by the type attribute. This property always returns submit. This property was added in JavaScript 1.1.
	value	Returns the value of the submit button specified by the value attribute.

Example

Listing 8.461 displays the use of the Submit properties. It contains a select box and a submit button. When the button is clicked, a second window is opened. The values of the properties of this Submit object are displayed in this window.

Listing 8.461 *Displaying the Properties of an Instance of a Submit Object*

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by clicking the button.
function openWin(){

    // Place the reference to the Submit button in a variable for easier access.
    var myInstance = document.myForm.mySubmit;

    // Open a window to store the results.
    var myWin = open("", "", "width=450,height=200");

    // Write the Submit button's properties to the window.
    myWin.document.write("The name is: " + myInstance.name + "<br>");
    myWin.document.write("The type is: " + myInstance.type + "<br>");

    // Note that the entire form object is passed with this property.
    // This allows you to then drill down and get the value of other
    // components in the form.
    myWin.document.write("The form can be used to grab the ");
    myWin.document.write("value of the button: ");
    myWin.document.write(myInstance.form.mySubmit.value);

    // Close the stream to the window.
    myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello, World!" name="myText">
    <input type="submit" value="Click to Submit" name="mySubmit"
        onclick="openWin()">
</form>
</body>
</html>

```

Submit.blur()

JavaScript1.0+, JScript1.0+
 Nav2+, IE3+, Opera3+

Syntax

```
submit.blur()
```

Description

The `blur()` method of the `Submit` object removes the focus from the submit button. Be careful when using this method in conjunction with the `Submit.focus()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.462 has a text box and a submit button. If the user clicks the Click Here to Remove Focus button, the button will no longer be focused. The best way to see this is to tab to the button and then click it. When tabbing to the button, you should see that it becomes selected.

Listing 8.462 Using the `blur()` Method to Remove the Focus from the Submit Button

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the removeFocus function called by pressing the button.
function removeFocus(){

    // Remove focus from the submit button.
    document.myForm.mySubmit.blur();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello, World!" name="myText">
    <input type="submit" value="Click Here to Remove Focus" name="mySubmit"
        onclick="removeFocus()">
</form>
</body>
</html>
```

Submit.click()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`submit.click()`

Description

The `click()` method of an instance of the `Submit` object simulates a click on the submit button. Note that if you have an `onClick` event handler assigned to this button, it will not be executed.

Example

Listing 8.463 has a text field, a submit button, and a link. If the user clicks the link, a function will be called to check whether the user entered any text. If the user did not, an alert box is displayed. If the user did enter text, the function submits the form by calling the `click()` method of the submit button.

Listing 8.463 *Using the `click()` Method to Submit a Form*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the submitForm function to submit the form.
function submitForm(){

    // Check to see if some text has been entered.
    if(document.myForm.myText.value == ""){
        alert("Please enter some text first");
    }else{

        // Use the click() method to submit the form.
        document.myForm.mySubmit.click();
    }
}
// -->
</script>
</head>
<body>
<form name="myForm">
Please Enter some text and click the link.<br>
    <input type="text" value="" name="myText">
    <input type="submit" value="Submit" name="mySubmit">
</form>
<br>
<a href="javascript:submitForm()">Click here to submit the form</a>
</body>
</html>
```

Submit.focus()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
submit.focus()
```

Description

The `focus()` method of the `Submit` object places focus on the `Submit` button. Be careful when using this method in conjunction with the `Submit.blur()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.464 has a text box and a submit button. If the user clicks in the text box, focus is placed on the submit button.

Listing 8.464 *Using the `focus()` Method to Set the Focus on the Submit Button*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the setFocus function called by clicking in the text box.
function setFocus(){

    // Place focus on the submit button.
    document.myForm.mySubmit.focus();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello, World!" name="myText" onfocus="setFocus()">
    <input type="submit" value="Submit" name="mySubmit">
</form>
</body>
</html>
```

Submit.form

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`submit.form`

Description

The `form` property of an instance of the `Submit` object provides access to all the data of the form in which the submit button is located.

Example

Listing 8.465 has a text box and a submit button. When the button is clicked, three properties of the form as a whole are displayed. These properties were referenced through the `form` property of the submit button.

Listing 8.465 *Accessing a Form Via the `form` Property of an Instance of a `Submit` Object*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the form property of submit button
    // in a variable for easier access.
    var formData = document.myForm.mySubmit.form;

    // Open a window to display the results.
    var myWin = open("", "", "width=450,height=200");

    // Write the form properties accessed through the form
    // property to the window.
    myWin.document.write("The name of the form is: " + formData.name + "<br>");
    myWin.document.write("The value of the text box is: ");
    myWin.document.write(formData.myText.value + "<br>");
    myWin.document.write("The name of the button is: ");
    myWin.document.write(formData.elements[1].name + "<br>");

    // Close the stream to the window.
    myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello, World!" name="myText">
    <input type="button" value="Click to Process" name="mySubmit"
        onclick="openWin()">
```

Listing 8.465 Continued

```
</form>
</body>
</html>
```

Submit.handleEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
submit.handleEvent(event)
```

Description

The `handleEvent()` method of the `Submit` object invokes the handler for the event specified. This method was added in JavaScript 1.2.

Example

Listing 8.466 has a single text box. The script tells the browser that it wants to intercept all `Click` events and that it wants the `myClickHandler` function to handle them. Within this function, the `handleEvent()` method of the submit button has been specified to handle the click.

When the user clicks anywhere on the page, the `onClick` event handler in the `<input type="submit">` tag calls a function to change the text in the text box. The change is nothing more than a simple number that is incremented, counting the number of times the page has been clicked.

Listing 8.466 Using the `handleEvent()` Method of a `Submit` Object to Handle all Clicks on a Page

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){
```

```
// Pass all click events to the onClick event of the submit button.
window.document.myForm.mySubmit.handleEvent(e);

}

// Function is called by onClick of the submit button. Displays the number
// of clicks that have occurred in the text box. Note that you have to
// return false so the form is not submitted.
function changeText(){
    counter++;
    document.myForm.myText.value = counter;
    return false;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onclick='changeText()'>
</form>
</body>
</html>
```

Submit.name

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

submit.name

Description

The name property of an instance of a Submit object returns the name of the submit button. This property is often accessed via the elements array of a Form object and used to return the name of the button. It is most useful when there are many forms on a given page, and determining the name helps you determine what function you want to perform.

Example

Listing 8.467 has a single text box and submit button. The elements array of a Form object is used to retrieve the name and display it in an alert box.

Listing 8.467 Using the name Property to Retrieve the Name of a Submit Button

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Display an alert box that contains the name of the
// submit button.
function getName(){
    alert("The name of this submit button is " +
        document.myForm.elements[1].name);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onclick='getName()'>
</form>
</body>
</html>
```

Submit.onBlur

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

onblur="command"

Description

The onBlur event handler is fired when the focus is moved away from that particular submit button. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the onFocus event handler or focus() method.

Example

Listing 8.468 contains two text boxes and a submit button. The button has the Blur event intercepted within its tag. The second text box is used as a counter for the number of times the onBlur event handler is fired.

In the script, the event calls a function that reassigns the submit button focus. The result of this is that when a user tries to click or tab away from the submit button box, the counter is incremented and the focus returns. As you will see if you run Listing 8.486, even clicking in other windows or the URL bar increments the counter.

Listing 8.468 Example of Using the `onBlur` Event Handler

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the submit button, and increment
// the counter in the text box.
function comeBack(){
    document.myForm.mySubmit.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='comeBack() '>
<form name="myForm">
    <input type="text" value="Text Box" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onblur='comeBack()'><br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>

```

Submit.onClick

JavaScript1.0+, JScript1.0+
 Nav2+, IE3+, Opera3+

Syntax

`onclick="command"`

Description

The `onClick` event handler is fired when a submit button is clicked. Note that this is not fired in the instances where a single form object is in the form, such as a text box, and the user presses the Enter or Return key to submit the form.

Example

Listing 8.469 has a single text box and a button. If the user presses the submit button, the `Click` event is fired, calling a function that changes the text in the text box to all uppercase.

Listing 8.469 Using the `onClick` Event Handler to Display the Contents of a Form Textbox Before It Is Submitted

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the setText function to change the text to uppercase.
function setText(){

    // Change the text to uppercase.
    document.myForm.myText.value = document.myForm.myText.value.toUpperCase();

}
// -->
</script>
</head>
<body>
<form name="myForm">
Please Enter some text and click the Submit Button.<br>
    <input type="text" value="Enter Text Here" name="myText">
    <input type="button" value="Submit" name="mySubmit" onclick="setText()">
</form>
</body>
</html>
```

Submit.onFocus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onfocus="command"`

Description

The `onFocus` event handler of an instance of a `Submit` object is fired when focus is set to that particular submit button. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onBlur` event handler or `blur()` method.

Example

Listing 8.470 contains two text boxes and a submit button. The submit button has the `Focus` event intercepted within its tag. The second text box is used for a counter for the number of times the `onFocus` event handler is fired.

In the script, the event calls a function that assigns the first text box focus. The result of this is that when a user tries to click or tab to the submit button, the counter is incremented and the focus is returned to the text box.

Listing 8.470 Example of Using the `onFocus` Event Handler

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the text box, and increment
// counter in text box.
function sendAway(){
    document.myForm.myText.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='sendAway() '>
<form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onfocus='sendAway()'><br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>

```

Submit.type

JavaScript1.1+, JScript1.0+
 Nav3+, IE3+, Opera3+

Syntax

`submit.type`

Description

The `type` property of an instance of a `Submit` object returns the type of the text box. This always returns `submit`.

Example

Listing 8.471 has a text box and a submit button. When the button is clicked, an alert box is popped up that displays the `type` property of the submit button.

Listing 8.471 Displaying the `type` Property in an Alert Box

```

<html>
<head>
<script language="JavaScript1.1" type="text/javascript">
<!--

```

Listing 8.471 Continued

```
// Display an alert box that contains the type of the
// submit button.
function getType(){
    alert("The name of this submit button is " +
        document.myForm.elements[1].type);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onclick='getType() '>
</form>
</body>
</html>
```

Submit.unwatch()

JavaScript 1.2+
Nav4+

Syntax

submit.unwatch(property)

Description

The *unwatch()* method of the Submit object is used to turn off the watch for a particular *property*.

Example

Listing 8.472 shows how the *unwatch()* method is used to stop watching the value property.

Listing 8.472 Example of the *unwatch()* Method

```
<html>
<head>
    <script type="text/javascript" language="JavaScript1.2">

    // Define a function to be called when the property changes.
    function alertme(id, oldValue, newValue){
        document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
        document.writeln(newValue + "<br>");
        return newValue;
    }
}
```



```
// Once loaded, watch the value property and then change it, and then
// turn off the watch.
function check(){
    document.myForm.mySubmit.watch("value", alertme);
    document.myForm.mySubmit.value="Send";
    document.myForm.mySubmit.unwatch("value");
}
</script>
</head>
<body onload="check()">
<form name="myForm">
    <input type="submit" name="mySubmit" value="Submit">
</form>
</body>
</html>
```

Submit.value

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

submit.value

Description

The value property of an instance of a Submit object returns the current value of the submit button. This value is what is displayed on the button itself.

Example

Listing 8.473 contains a text box and button. When a user clicks the submit button, an alert box pops up displaying the value of this instance.

Listing 8.473 *Accessing the Value of a Submit Object*

```
<html>
<head>
<script language="JavaScript1.1" type="text/javascript">
<!--

// Display an alert box that contains the value of the
// submit button.
function getValue(){
    alert("The value of this submit button is " +
        document.myForm.elements[1].value);
}
```

Listing 8.473 Continued

```
// -->
</script>
</head>
<body>
<form name="myForm">
  <input type="text" value="First Box" name="myText">
  <input type="submit" value="Submit" name="mySubmit" onclick='getValue() '>
</form>
</body>
</html>
```

Submit.watch()

JavaScript 1.2+
Nav4+

Syntax

submit.watch(property, function)

Description

The `watch()` method of the `Submit` object is used to turn on the watch for a particular *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 8.474 shows how the `watch()` method is used to start watching the `value` property.

Listing 8.474 Example of the `watch()` Method

```
<html>
<head>
  <script type="text/javascript" language="JavaScript1.2">

    // Define a function to be called when the property changes.
    function alertme(id, oldValue, newValue){
      document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
      document.writeln(newValue + "<br>");
      return newValue;
    }

    // Once page is loaded, watch the value property then change it.
    function check(){
      document.myForm.mySubmit.watch("value", alertme);
      document.myForm.mySubmit.value="Send";
    }
  </script>
</head>
<body>
  <form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="submit" value="Submit" name="mySubmit" onclick='check() '>
  </form>
</body>
</html>
```

```

    </script>
</head>
<body onload="check()">
<form name="myForm">
    <input type="submit" name="mySubmit" value="Submit">
</form>
</body>
</html>

```

taint()

JavaScript1.1
Nav3

Syntax

taint(*object*)

Description

The taint() method was a security measure that was only implemented in JavaScript 1.1 and that allowed a developer to keep return values from being used by and propagated to other scripts. This method does not change the data element passed to it, but rather returns a marked reference to the element.

Because taint() and the functionality of data tainting was removed in JavaScript 1.2, you should avoid using this method. You should use it only if you have a specific security reason for compatibility with Navigator 3 browsers. See Chapter 1, “What is JavaScript to a Programmer?” for more information on the security model that is now used in Navigator browsers.

Example

Listing 8.475 simply taints a variable that is defined in a separate window.

Listing 8.475 Use of the taint() Method, Which Is No Longer Supported

```

<script language="JavaScript1.1" type="text/javascript">
<!--

// Store the tainted variable from the second window
// in the variable taintMyVar.
var taintMyvar = taint(myWin.myVar);

// -->
</script>

```

Text

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Text object is one of the core JavaScript objects. Instances are created by the browser when it encounters an HTML `<input>` tag with the `type` attribute set to `text`. In the JavaScript object hierarchy, the Text object is located at `window.document.Form.Text`. Table 8.37 lists the properties, methods, and events used by the Text object.

Table 8.37 *Event Handlers, Methods, and Properties Used by the Text Object*

Type	Item	Description
Event Handler	onBlur	Executes code when the text box loses the focus.
	onChange	Executes code when the text box loses the focus and has had its value modified.
	onFocus	Executes code when the text box receives the focus.
	onSelect	Executes code when a user selects some of the text within the text box.
Method	blur()	Removes the focus from the text box.
	focus()	Gives the focus to the text box.
	handleEvent()	Invokes the handler for the event specified and was added in JavaScript 1.2.
	select()	Selects the text in the text box.
	unwatch()	Used to turn off the watch for a particular property.
	watch()	Used to turn on the watch for a particular property.
Property	defaultValue	Returns the value of the text box specified by the <code>value</code> attribute. Note that this property is not supported by the Opera browsers.
	form	Returns the entire form the text box is in.
	name	Returns the name of the text box specified by the <code>name</code> attribute.
	type	Returns the type of the text box specified by the <code>type</code> attribute. Note that this is always <code>text</code> and was added in JavaScript 1.1.
	value	Returns the value that is actually displayed in the text box.

Example

Listing 8.476 displays the use of the Text properties. It contains a text box and a button. When the button is clicked, a second window is opened. The values of the properties of this Text object are displayed in the second window.

Listing 8.476 *Displaying the Properties of an Instance of a Text Object*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the text box in a variable for easier access.
    var myInstance = document.myForm.myText;

    // Open a window to store the results.
    var myWin = open("", "", "width=450,height=200");

    // Write the text box's properties to the window.
    myWin.document.write("The defaultValue is: " + myInstance.defaultValue);
    myWin.document.write("<br>");
    myWin.document.write("The name is: " + myInstance.name + "<br>");
    myWin.document.write("The type is: " + myInstance.type + "<br>");
    myWin.document.write("The value is: " + myInstance.value + "<br>");

    // Note that the entire form object is passed with this property.
    // This allows you to then drill down and get the value of other
    // components in the form.
    myWin.document.write("The form can be used to grab the value ");
    myWin.document.write("of the button: ");
    myWin.document.write(myInstance.form.myButton.value);

    // Close the stream to the window.
    myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello world" name="myText">
    <input type="button" value="Click to Process" name="myButton"
        onclick="openWin()">
</form>
</body>
</html>
```

Text.blur()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
text.blur()
```

Description

The `blur()` method of the `Text` object removes focus from the text box. Be careful when using this method in conjunction with the `Text.focus()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.477 has a text box and a button. If the user highlights some of the text in the box and then clicks the Click Here to Remove Focus button, the text will no longer be highlighted.

Listing 8.477 Using the `blur()` Method to Remove Focus from the Text Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the removeFocus function called by pressing the button.
function removeFocus(){

    // Remove focus from the text box.
    document.myForm.myText.blur();
}
// -->
</script>
</head>
<body>
<b>Highlight some of the text in the following text box:</b>
<p>
<form name="myForm">
    <input type="text" value="hello world" name="myText">
    <input type="button" value="Click Here to Remove Focus" name="myButton"
        onclick="removeFocus()">
</form>
</body>
</html>
```

Text.defaultValue

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera5+

Syntax

`text.defaultValue`

Description

The `defaultValue` property of a `Text` object instance contains the default value specified by the `value` attribute of the `<input>` tag. This property is often used to reset forms to their default values after a user has entered some data.

Example

Listing 8.478 has a text box and a button. If the user edits some of the text in the box and then clicks the Click to Reset button, the text will change back to the default value.

Listing 8.478 Using the `defaultValue` Property to Set the Value of the Text Box Back to Its Original Value

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the resetForm function called by pressing the button.
function resetForm(){

    // Set the text in the text box back to "hello world".
    document.myForm.myText.value = document.myForm.myText.defaultValue;
}
// -->
</script>
</head>
<body>
<b>Edit the text in the following text box:</b>
<p>
<form name="myForm">
    <input type="text" value="hello world" name="myText">
    <input type="button" value="Click to Reset" name="myButton"
        onclick="resetForm()">
</form>
</body>
</html>
```

Text.focus()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
text.focus()
```

Description

The `focus()` method of the `Text` object gives focus to the text box. Be careful when using this method in conjunction with the `Text.blur()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.479 has two text boxes and two buttons. If the user clicks the first button, the cursor is set inside the first text box. If the user clicks the second button, the cursor is set inside the second text box.

Listing 8.479 Using the `focus()` Method to Place the Cursor in the Desired Text Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the setFocus function called by pressing the button.
function setFocus(num){

    // Determine which button was clicked and set the cursor
    // in the appropriate text box.
    if(num == 1){
        document.myForm.myText1.focus();
    }else if(num == 2){
        document.myForm.myText2.focus();
    }
}
// -->
</script>
</head>
<body>
<p>
<form name="myForm">
    <input type="text" value="Textbox 1" name="myText1">
    <input type="button" value="Click to Set Cursor" name="myButton1"
        onclick="setFocus(1)">
    <br>
```



```



```

Text.form

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`text.form`

Description

The `form` property of an instance of a `Text` object holds all the data of the form in which the text box is contained. This allows a developer to obtain specific information about the form in which the text box is located.

Example

Listing 8.480 has a text box and a button. When the button is clicked, three properties of the form as a whole are displayed. These properties were referenced through the `form` property of the text box.

Listing 8.480 *Accessing a Form via the `form` Property of an Instance of a `Text` Object*

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the form property of the text box
    // in a variable for easier access.
    var formData = document.myForm.myText.form;

    // Open a window to display the results.
    var myWin = open("", "", "width=450,height=200");

    // Write the form properties accessed through the form
    // property to the window.
    myWin.document.write("The name of the form is: " + formData.name + "<br>");
    myWin.document.write("The value of the text box is: ");

```

Listing 8.480 Continued

```
myWin.document.write(formData.myText.value + "<br>");
myWin.document.write("The name of the button is: ");
myWin.document.write(formData.elements[1].name + "<br>");

// Close the stream to the window.
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
  <input type="text" value="Hello world" name="myText">
  <input type="button" value="Click to Process" name="myButton"
    onclick="openWin()">
</form>
</body>
</html>
```

Text.handleEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
text.handleEvent(event)
```

Description

The `handleEvent()` method of the `Text` object invokes the handler for the event specified. This method was added in JavaScript 1.2.

Example

Listing 8.481 has a single text box. The script tells the browser that it wants to intercept all `Click` events and that it wants the `myClickHandler` function to handle them. Within this function, the `handleEvent()` method of the text box has been specified to handle the click.

When the user clicks anywhere on the page, the `onClick` event handler in the `<input type="text">` tag calls a function to change the text in the text box. The change is nothing more than a simple number that is incremented, counting the number of times the page has been clicked.

Listing 8.481 Using the `handleEvent()` Method of a `Text` Object to Handle All Clicks on a Page

```
<html>
<head>
```

```

<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text box.
    window.document.myForm.myText.handleEvent(e);

}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText" onclick='changeText()''>
</form>
</body>
</html>

```

Text.name

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`text.name`

Description

The name property of an instance of a Text object returns the name of the text box. This property is often accessed via the elements array of a Form object and used to return the name of the text area. It is most useful when there are many forms on a given page, and determining the name helps you determine what function you want to perform.

Example

Listing 8.482 has a single text box and button. The `elements` array of a `Form` object is used to retrieve the name and display it in an alert box.

Listing 8.482 *Using the `name` Property to Retrieve the Name of a Text Box*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Display an alert box that contains the name of the
// text box.
function getName(){
    alert("The name of this text box is " +
        document.myForm.elements[0].name);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="button" value="Get Name" name="myButton" onclick='getName()'>
</form>
</body>
</html>
```

Text.onBlur

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onblur="command"`

Description

The `onBlur` event handler of an instance of a `Text` object is fired when the focus is moved away from that particular text box. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onFocus` event handler or the `focus()` method.

Example

Listing 8.483 contains three text boxes, one of which has the `onBlur` event handler intercepted within its tag. The third text box is used for a counter for the number of times the `onBlur` event handler is fired.

In the script, the event calls a function that reassigns the first text box focus. The result of this is that when a user tries to click or tab away from the first text box, the counter is incremented and the focus returns. As you will see if you run Listing 8.499, even clicking in other windows or the URL bar increments the counter.

Listing 8.483 Example of Using the `onBlur` Event Handler

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the first text box, and increment
// counter in last text box.
function comeBack(){
    document.myForm.myText1.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='comeBack() '>
<form name="myForm">
    <input type="text" value="First Box" name="myText1" onblur='comeBack()' '>
    <input type="text" value="Second Box" name="myText2"><br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>
```

Text.onChange

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onChange="command"`

Description

The `onChange` event handler of an instance of a `Text` object is fired when the text in the box is modified. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when focus is placed on or away from the text box.

Example

Listing 8.484 has a single text box. If the user changes the text and then shifts the focus away from the text box by clicking elsewhere or pressing Return, the default text is placed back in the text box.

Listing 8.484 *Using the `onChange` Event Handler to Change the Text Back to the Default*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Change the text back to the default if user tries to change it.
// Note that the user has to click away or hit return for this
// to change back.
function changeBack(){
    document.myForm.myText.value = document.myForm.myText.defaultValue;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Change Me?" name="myText" onchange='changeBack() '>
</form>
</body>
</html>
```

Text.onFocus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onfocus="command"
```

Description

The `onFocus` event handler of an instance of a `Text` object is fired when focus is made on that particular text box. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onBlur` event handler or the `blur()` method.

Example

Listing 8.485 contains three text boxes, one of which has the `onFocus` event handler intercepted within its tag. The third text box is used for a counter for the number of times the Focus event is fired.

In the script, the event handler calls a function that assigns the second text box focus. The result of this is that when a user tries to click or tab to the first text box, the counter is incremented and the focus is returned to the second text the box.

Listing 8.485 Example of Using the `onFocus` Event Handler

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the second text box, and increment
// counter in last text box.
function sendAway(){
    document.myForm.myText2.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='sendAway() '>
<form name="myForm">
    <input type="text" value="First Box" name="myText1" onfocus='sendAway()' '>
    <input type="text" value="Second Box" name="myText2"><br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>
```

Text.onSelect

JavaScript1.0+, JScript1.0+
IE3+, Net6

Syntax

`onselect="command"`

Description

The `onSelect` event handler of an instance of a `Text` object is fired when the text in the box is highlighted. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when focus is placed on the text box.

NOTE

Note that Netscape defined this function in JavaScript 1.0; however, it was not fully implemented until Netscape 6. Opera browsers do not support this function either.

Example

Listing 8.486 has two text boxes. If the user highlights the text in the first text box, the default text of the first text box is written to the second text box.

Listing 8.486 Using the onSelect Event Handler to Set the Text in the Second Text Box

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Change the text of the second text box to the default
// of the first if user highlights text in the first.
function setText(){
    document.myForm.myText2.value = document.myForm.myText1.defaultValue;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Change Me?" name="myText1" onselect='setText()'>
    <br>
    <input type="text" value="" name="myText2">
</form>
</body>
</html>
```

Text.select()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
text.select()
```

Description

The `select()` method of the `Text` object selects the text in the text box. Be careful when using this method in conjunction with the `blur()` and `focus()` methods. It can lead to a focus/blur loop where the browser blurs or focuses as soon as it has been selected, and vice versa.

Example

Listing 8.487 has a text box and a button. If the user clicks the button, the text inside the text box will be highlighted. Notice that the `focus()` method had to be used to tell the browser to actually highlight the text.

Listing 8.487 *Using the `select()` Method to Select the Text in a Text Box*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the selectText function called by pressing the button.
function selectText(){

    // Select the text in the box, then place focus on it.
    document.myForm.myText.select();
    document.myForm.myText.focus();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="Hello world" name="myText">
    <input type="button" value="Click to Select Text" name="myButton"
        onclick="selectText()">
</form>
</body>
</html>
```

Text.type

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

`text.type`

Description

The `type` property of an instance of a `Text` object returns the type of the text box. This always returns `text`.

Example

Listing 8.488 has a text box and a button. When the button is clicked, an alert box is popped up that displays the `type` property of the text box.

Listing 8.488 Displaying the type Property in an Alert Box

```
<html>
<head>
<script language="JavaScript1.1" type="text/javascript">
<!--

// Display an alert box that contains the type of the
// text box.
function getType(){
    alert("The name of this text box is " +
        document.myForm.elements[0].type);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" value="First Box" name="myText">
    <input type="button" value="Get Type" name="myButton" onclick='getType()'>
</form>
</body>
</html>
```

Text.unwatch()

JavaScript 1.2+
Nav4+

Syntax

`text.unwatch(property)`

Description

The `unwatch()` method of the `text` object is used to turn off the watch for a particular *property*.

Example

Listing 8.489 shows how the `unwatch()` method is used to stop watching the value property.

Listing 8.489 Example of the `unwatch()` Method

```
<html>
<head>
<script type="text/javascript" language="JavaScript1.2">

function alertme(id, oldValue, newValue){
    document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
    document.writeln(newValue + "<br>");
}
```

```

    return newValue;
}

function check(){
    document.myForm.myItem.watch("value", alertme);
    document.myForm.myItem.unwatch("value");
    document.myForm.myItem.value="Send";
}
</script>
</head>
<body onload="check()">
<form name="myForm">
    <input type="text" name="myItem" value="Submit">
</form>
</body>
</html>

```

Text.value

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`text.value`

Description

The `value` property of an instance of a `Text` object returns the current value of the text box. Note that this is not the default value that can be accessed via the `Text.defaultValue` property and is often used to set the value of a text box.

Example

Listing 8.490 contains a text box and button. You can edit the text in the text box and then click the Reset button to reset the text back to the default value.

Listing 8.490 Resetting the Value of a Text Box to the Default Value

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Reset the text in the text box to its default value.
function resetText(){
    document.myForm.myText.value = document.myForm.myText.defaultValue;
}
// -->
</script>
</head>

```

Listing 8.490 Continued

```
<body>
<form name="myForm">
  <input type="text" value="First Box" name="myText">
  <input type="button" value="Reset" name="myButton" onclick='resetText()'>
</form>
</body>
</html>
```

Text.watch()

JavaScript 1.2+
Nav4+

Syntax

text.watch(*property*, *function*)

Description

The watch() method of the Text object is used to turn on the watch for a particular *property*. Any time the specified *property* is changed after the watch() method has been called, the specified *function* is called.

Example

Listing 8.491 shows how the watch() method is used to start watching the value property.

Listing 8.491 Example of the watch() Method

```
<html>
<head>
  <script type="text/javascript" language="JavaScript1.2">

    function alertme(id, oldValue, newValue){
      document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
      document.writeln(newValue + "<br>");
      return newValue;
    }

    function check(){
      document.myForm.myItem.watch("value", alertme);
      document.myForm.myItem.value="Send";
    }
  </script>
</head>
<body onload="check()">
<form name="myForm">
  <input type="text" name="myItem" value="Submit">
```

```

</form>
</body>
</html>

```

Textarea

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Textarea object is one of the core JavaScript objects. Instances are created by the browser when it encounters an HTML `<textarea>` tag. In the JavaScript object hierarchy, the Textarea object is located at `window.document.Form.Textarea`. Table 8.38 lists the event handlers, methods, and properties used by the Textarea object.

Table 8.38 Event Handlers, Methods, and Properties Used by the Textarea Object

Type	Item	Description
Event Handler	<code>onBlur</code>	Executes code when the text area loses the focus.
	<code>onChange</code>	Executes code when the text area loses the focus and has had its value modified.
	<code>onFocus</code>	Executes code when the text area receives the focus.
	<code>onKeyDown</code>	Executes code when a key is pressed down. This occurs before an <code>onKeyPress</code> event handler is triggered and was added in JavaScript 1.2.
	<code>onKeyPress</code>	Executes code when a key is pressed down immediately after an <code>onKeyDown</code> event handler is triggered. This event handler was added in JavaScript 1.2.
	<code>onKeyUp</code>	Executes code when a key is released. This was added in JavaScript 1.2.
	<code>onSelect</code>	Executes code when a user selects some of the text within the text area.
Method	<code>blur()</code>	Removes the focus from the text area.
	<code>focus()</code>	Gives the focus to the text area.
	<code>handleEvent()</code>	Invokes the handler for the event specified and was added in JavaScript 1.2.
	<code>select()</code>	Selects the text in the text area.
	<code>unwatch()</code>	Used to turn off the watch for a particular property.
	<code>watch()</code>	Used to turn on the watch for a particular property.

Table 8.38 Continued

Type	Item	Description
Property	defaultValue	Returns the value of the text area defined between the beginning and ending <textarea> tags. Note that this property is not supported by the Opera browsers.
	form	Returns the entire form the text area is in.
	name	Returns the name of this text area specified by the name attribute.
	type	Returns the type of this text area. Note that this is always textarea and was added in JavaScript 1.1.
	value	Returns the value that is actually displayed in the text area.

Example

Listing 8.492 displays the use of the Textarea properties. It contains a text area and a button. When the button is clicked, a second window is opened. The values of the properties of this Textarea object are displayed in this window.

Listing 8.492 *Displaying the Properties of an Instance of a Textarea Object*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the text area in a variable for easier access.
    var myInstance = document.myForm.myTextArea;

    // Open a window to store the results
    var myWin = open("", "", "width=450,height=200");

    // Write the text area's properties to the window.
    myWin.document.write("The defaultValue is: ");
    myWin.document.write(myInstance.defaultValue + "<br>");
    myWin.document.write("The name is: " + myInstance.name + "<br>");
    myWin.document.write("The type is: " + myInstance.type + "<br>");
    myWin.document.write("The value is: " + myInstance.value + "<br>");

    // Note that the entire form object is passed with this property.
    // This allows you to then drill down and get the value of other
    // components in the form.
    myWin.document.write("The form can be used to grab the ");
```

```

myWin.document.write("value of the button: ");
myWin.document.write(myInstance.form.myButton.value);

// Close the stream to the window.
myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
  <textarea name="myTextArea" rows="6" cols="50">
    Here is some text in my text area.
  </textarea>
  <input type="button" value="Click to Process" name="myButton"
    onclick="openWin()">
</form>
</body>
</html>

```

Textarea.blur()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
textarea.blur()
```

Description

The `blur()` method of the `Textarea` object removes the focus from the text area. Be careful when using this method in conjunction with the `Textarea.focus()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.493 has a text area and a button. If the user highlights some of the text in the box and then clicks the Click Here to Remove Focus button, the text will no longer be highlighted.

Listing 8.493 *Using the `blur()` Method to Remove the Focus from the Text Area*

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

```

Listing 8.493 Continued

```
// Define the removeFocus function called by pressing the button.
function removeFocus(){

    // Remove focus from the text area.
    document.myForm.myTextArea.blur();
}
// -->
</script>
</head>
<body>
<b>Highlight some of the text in the following text area:</b>
<p>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
    <input type="button" value="Click Here to Remove Focus" name="myButton"
        onclick="removeFocus()">
</form>
</body>
</html>
```

Textarea.defaultValue

JavaScript1.0+, JScript1.0+
Nav2+, IE3+

Syntax

`textarea.defaultValue`

Description

The `defaultValue` property of a `Textarea` object instance contains the text between the beginning and ending `<textarea>` tags. This property is often used to reset areas to their default values after a user has modified them.

Example

Listing 8.494 has a text area and a button. If the user edits some of the text in the box and then clicks the Click to Reset button, the text will change back to the default value.

Listing 8.494 Using the `defaultValue` Property to Set the Value of the Text Area Back to Its Original Value

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--
```



```
// Define the resetForm function called by pressing the button.
function resetForm(){

    // Set the text in the text area back to "hello world".
    document.myForm.myTextArea.value = document.myForm.myTextArea.defaultValue;
}
// -->
</script>
</head>
<body>
<b>Edit the text in the following text box:</b>
<p>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
<input type="button" value="Click to Reset" name="myButton"
    onclick="resetForm()">
</form>
</body>
</html>
```

Textarea.focus()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
textarea.focus()
```

Description

The `focus()` method of the `Textarea` object gives the focus to the text area. Be careful when using this method in conjunction with the `Textarea.blur()` method. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.495 has two text areas and two buttons. If the user clicks the first button, the cursor is set inside the first text area. If the user clicks the second button, the cursor is set inside the second text area.

Listing 8.495 Using the `focus()` Method to Place the Cursor in the Desired Text Area

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
```

Listing 8.495 Continued

```
<!--  
  
// Define the setFocus function called by pressing the button.  
function setFocus(num){  
  
    // Determine which button was clicked and set the cursor  
    // in the appropriate text area.  
    if(num == 1){  
        document.myForm.myTextArea1.focus();  
    }else if(num == 2){  
        document.myForm.myTextArea2.focus();  
    }  
}  
// -->  
</script>  
</head>  
<body>  
<p>  
<form name="myForm">  
    <textarea name="myTextArea1" rows="2" cols="50">  
        Here is the first text area.  
    </textarea>  
<input type="button" value="Click to Set Cursor" name="myButton1"  
        onclick="setFocus(1)">  
    <br>  
    <textarea name="myTextArea2" rows="2" cols="50">  
        Here is the second text area.  
    </textarea>  
    <input type="button" value="Click to Set Cursor" name="myButton2"  
        onclick="setFocus(2)">  
</form>  
</body>  
</html>
```

Textarea.form

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`textarea.form`

Description

The `form` property of an instance of a `Textarea` object holds all the data of the form in which the text box is contained. This allows a developer to obtain specific information about the form in which the text area is located.

Example

Listing 8.496 has a text area and a button. When the button is clicked, three properties of the form as a whole are displayed. These properties were referenced through the form property of the text area.

Listing 8.496 *Accessing a Form via the form Property of an Instance of a Textarea Object*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin function called by pressing the button.
function openWin(){

    // Place the reference to the form property of text area
    // in a variable for easier access.
    var formData = document.myForm.myTextArea.form;

    // Open a window to display the results.
    var myWin = open("", "", "width=450,height=200");

    // Write the form properties accessed through the form
    // property to the window.
    myWin.document.write("The name of the form is: " + formData.name + "<br>");
    myWin.document.write("The value of the text box is: ");
    myWin.document.write(formData.myTextArea.value + "<br>");
    myWin.document.write("The name of the button is: ");
    myWin.document.write(formData.elements[1].name + "<br>");

    // Close the stream to the window.
    myWin.document.close();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
    <input type="button" value="Click to Process" name="myButton"
        onclick="openWin()">
</form>
</body>
</html>
```

Textarea.handleEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
textarea.handleEvent(event)
```

Description

The `handleEvent()` method of the `Textarea` object invokes the handler for the event specified. This method was added in JavaScript 1.2.

Example

Listing 8.497 has a single text area. The script tells the browser that it wants to intercept all `Click` events and that it wants the `myClickHandler` function to handle them. Within this function, the `handleEvent()` method of the text area has been specified to handle the click.

When the user clicks anywhere on the page, the `onClick` event handler in the `<textarea>` tag calls a function to change the text in the text area. The change is nothing more than a simple number that is incremented, counting the number of times the page has been clicked.

Listing 8.497 Using the `handleEvent()` Method of a `Textarea` Object to Handle All Clicks on a Page

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text area.
    window.document.myForm.myTextArea.handleEvent(e);

}
```

```
// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myTextArea.value = counter++;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50" onclick='changeText()''>
        Here is some text in my text area.
    </textarea>
</form>
</body>
</html>
```

Textarea.name

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`textarea.name`

Description

The name property of an instance of a Textarea object returns the name of the text area. This property is often accessed via the elements array of a Form object and used to return the name of the text area. It is most useful when there are many forms on a given page, and determining the name helps you determine what function you want to perform.

Example

Listing 8.498 has a text area and button. The elements array of a Form object is used to retrieve the name and display it in an alert box.

Listing 8.498 *Using the name Property to Retrieve the Name of a Text Area*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Display an alert box that contains the name of the
// text area.
```

Listing 8.498 Continued

```
function getName(){
    alert("The name of this text area is " +
        document.myForm.elements[0].name);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
    <input type="button" value="Get Name" name="myButton" onclick='getName()'/>
</form>
</body>
</html>
```

Textarea.onBlur

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onblur="command"
```

Description

The `onBlur` event handler of an instance of a `Textarea` object is fired when the focus is moved away from that particular text area. Care should be taken when using this event handler because it is possible to get into an infinite loop when using `onFocus` event handler or `focus()` method.

Example

Listing 8.499 contains three text areas, one of which has the `onBlur` event handler used within its tag. The third text box is used as a counter for the number of times the `onBlur` event handler is used.

In the script, the event calls a function that reassigns the first text area focus. The result of this is that when a user tries to click or tab away from the first text area, the counter is incremented and the focus returns. As you will see if you run Listing 8.513, even clicking in other windows or the URL bar increments the counter.

Listing 8.499 Example of Using the `onBlur` Event Handler

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
```

```

<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the first text area, and increment
// the counter in last text area.
function comeBack(){
    document.myForm.myTextArea1.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='comeBack() '>
<form name="myForm">
    <textarea name="myTextArea1" rows="2" cols="50" onblur='comeBack()' '>
        Here is some text in my text area.
    </textarea>
    <textarea name="myTextArea2" rows="2" cols="50">
        Here is some text in my text area.
    </textarea><br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>

```

Textarea.onChange

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

onChange="command"

Description

The onChange event handler of an instance of a Textarea object is fired when the text in the area is modified. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when focus is placed on or away from the text area.

Example

Listing 8.500 has a single text area. If the user changes the text and then shifts the focus away from the text area by clicking elsewhere, the default text is placed back in the text area.

Listing 8.500 Using the onChange Event Handler to Change the Text Back to the Default

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Change the text back to the default if user tries to change it.
// Note that the user has to click away or hit return for this
// to change back.
function changeBack(){
    document.myForm.myTextArea.value = document.myForm.myTextArea.defaultValue;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="2" cols="50" onchange='changeBack()'>
        Here is some text in my text area.
    </textarea>
</form>
</body>
</html>
```

Textarea.onFocus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onfocus="command"
```

Description

The onFocus event handler of an instance of a Textarea object is fired when focus is made on that particular text area. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the onBlur event handler or the blur() method.

Example

Listing 8.501 contains three text boxes, one of which has the Focus event intercepted within its tag. The third text box is used for a counter for the number of times the Focus event is fired.

In the script, the event calls a function that assigns the second text box focus. The result of this is that when a user tries to click or tab to the first text box, the counter is incremented and the focus is returned to the second text box.

Listing 8.501 Example of Using the `onFocus` Event Handler

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Initialize a counter to show clicks.
var counter = 0;

// Set the focus on the second text area, and increment
// the counter in last text area.
function sendAway(){
    document.myForm.myTextArea2.focus();
    document.myForm.counter.value = counter++;
}
// -->
</script>
</head>
<body onload='sendAway() '>
<form name="myForm">
    <textarea name="myTextArea1" rows="2" cols="50" onfocus='sendAway()' '>
    Here is some text in my text area.
    </textarea>
    <textarea name="myTextArea2" rows="2" cols="50">
    Here is some text in my text area.
    </textarea>
    <br>
    <input type="text" size="2" value="" name="counter">
</form>
</body>
</html>

```

Textarea.onKeyDown

JavaScript1.2+, JScript3.0+
 Nav4+, IE4+

Syntax

`onkeydown="command"`

Description

The `onKeyDown` event handler of an instance of a `Textarea` object is fired when a key is pressed down within the text area. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when the focus is placed on or away from the text area or other key-related events are used. The `onKeyDown` event handler is called followed by a `onKeyPress` event handler.

Example

Listing 8.502 has a single text area. If the user presses a key while the focus is on the text area, an alert box is displayed as soon as the key is pressed.

Listing 8.502 The onKeyDown Event Handler Causes an Alert Box to Be Displayed

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Pop up an alert box when the user presses a key.
function showDialog(){
    alert("A key was pressed down");
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="2" cols="50" onkeydown='showDialog()'>
        Here is some text in my text area.
    </textarea>
</form>
</body>
</html>
```

Textarea.onKeyPress

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

onkeypress="command"

Description

The onKeyPress event handler of an instance of a Textarea object is fired when a key is pressed within the text area. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when the focus is placed on or away from the text area or other key-related events are used. This event is called after an onKeyDown event handler.

Example

Listing 8.503 has a single text area. If the user presses a key while the focus is on the text area, an alert box is displayed as soon as the key is pressed, showing the type of event that was fired. The first event handler is onKeyDown, which is followed by an onKeyPress event handler.

Listing 8.503 The onKeyDown and onKeyPress Event Handlers Cause an Alert Box to Be Displayed

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Pop up an alert box when the user presses a key.
function showDialog(type){
    alert("An onKeyDown + type + " event just occurred");
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="2" cols="50"
        onkeydown='showDialog("Down")'
        onkeypress='showDialog("Press")'>
        Here is some text in my text area.
    </textarea>
</form>
</body>
</html>
```

Textarea.onKeyUp

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

onkeyup="command"

Description

The onKeyUp event handler of an instance of a Textarea object is fired when a key is released within the text area. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when the focus is placed on or away from the text area or other key-related events are used. The onKeyUp event handler is called after an onKeyPress event handler.

Example

Listing 8.504 has a single text area. If the user releases a key while the focus is on the text area, an alert box is displayed as soon as the key is released.

Listing 8.504 *The onKeyUp Event Handler Causes an Alert Box to Be Displayed*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Pop up an alert box when the user releases a key.
function showDialog(){
    alert("A key was released");
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="2" cols="50" onkeyup='showDialog()''>
        Here is some text in my text area.
    </textarea>
</form>
</body>
</html>
```

Textarea.onSelect

JavaScript1.0+, JScript1.0+
IE3+, Net6

Syntax

onselect="command"

Description

The onSelect event handler of an instance of a Textarea object is fired when the text in the area is highlighted. Care should be taken when using this event handler because it is possible to get into an infinite loop when using other events or methods that are fired when the focus is placed on the text area.

NOTE

Note that Netscape defined this function in JavaScript 1.0; however, it was not fully implemented until Netscape 6 was released. Opera browsers do not support this function either.

Example

Listing 8.505 has two text areas. If the user highlights the text in the first text area, the default text of the first text area is written to the second text area.

Listing 8.505 Using the onSelect Event Handler to Set the Text in the Second Text Box

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Change the text of the second text area to the default
// of the first if user highlights text in the first.
function setText(){
    document.myForm.myTextArea2.value = document.myForm.myTextArea1.defaultValue;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea1" rows="2" cols="50" onselect='setText()'>
        Here is some text in my text area.
    </textarea>
    <br>
    <textarea name="myTextArea2" rows="2" cols="50">
    </textarea>
</form>
</body>
</html>

```

Textarea.select()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
textarea.select()
```

Description

The `select()` method of the `Textarea` object selects the text in the text area. Be careful when using this method in conjunction with the `blur()` and `focus()` methods. It can lead to a focus/blur loop where the browser blurs or focuses as soon as it has been selected, and vice versa.

Example

Listing 8.506 has a text area and a button. If the user clicks the button, the text inside the text area will be highlighted. Notice the `focus()` method had to be used to tell the browser to actually highlight the text.

Listing 8.506 *Using the select() Method to Select the Text in a Text Area*

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the selectText function called by pressing the button.
function selectText(){

    // Select the text in the area, then place focus on it.
    document.myForm.myTextArea.select();
    document.myForm.myTextArea.focus();
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
    <input type="button" value="Click to Select Text" name="myButton"
        onclick="selectText()">
</form>
</body>
</html>
```

Textarea.type

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

`textarea.type`

Description

The type property of an instance of a Textarea object returns the type of the text area. This always returns `textarea`.

Example

Listing 8.507 has a text area and a button. When the button is clicked, an alert box is popped up that displays the type property of the text area.

Listing 8.507 *Displaying the type Property in an Alert Box*

```
<html>
<head>
```

```

<script language="JavaScript1.1" type="text/javascript">
<!--

// Display an alert box that contains the type of the
// text area.
function getType(){
    alert("The name of this text area is " +
        document.myForm.elements[0].type);
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <textarea name="myTextArea" rows="6" cols="50">
        Here is some text in my text area.
    </textarea>
    <input type="button" value="Get Type" name="myButton" onclick='getType() '>
</form>
</body>
</html>

```

Textarea.unwatch()

JavaScript 1.2+
Nav4+

Syntax

textarea.unwatch(property)

Description

The `unwatch()` method of the `Textarea` object is used to turn off the watch for a particular *property*.

Example

Listing 8.508 shows how the `unwatch()` method is used to stop watching the value property.

Listing 8.508 Example of the `unwatch()` Method

```

<html>
<head>
    <script type="text/javascript" language="JavaScript1.2">

        // Write out a note if the value of the Textarea box changes.
        function alertme(id, oldValue, newValue){
            document.writeln("ID (" + id + ") changed from " + oldValue + " to ");
            document.writeln(newValue + "<br>");
        }
    </script>

```

Listing 8.508 Continued

```
        return newValue;
    }

    // Watch the value of the Teaxtarea box, then change it, before unwatching.
    function check(){
        document.myForm.myItem.watch("value", alertme);
        document.myForm.myItem.value="Send";
        document.myForm.myItem.unwatch("value");
    }
</script>
</head>
<body onload="check()">
<form name="myForm">
    <textarea name="myItem" rows="3" cols="30">Submit</textarea>
</form>
</body>
</html>
```

Textarea.value

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`textarea.value`

Description

The value property of an instance of a Textarea object returns the current value of the text area. Note that this is not the default value that can be accessed via the Textarea.defaultValue property and is often used to set the value of a text area.

Example

Listing 8.509 contains a text area and button. You can edit the text in the text area, and then click the Reset button to reset the text back to the default value.

Listing 8.509 Resetting the Value of a Text Area to the Default Value

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Reset the text in the text area to its default value.
function resetText(){
    document.myForm.myTextArea.value = document.myForm.myTextArea.defaultValue;
}
// -->
```



```

</script>
</head>
<body>
<form name="myForm">
  <textarea name="myTextArea" rows="6" cols="50">
    Here is some text in my text area.
  </textarea>
  <input type="button" value="Reset" name="myButton" onclick='resetText()''>
</form>
</body>
</html>

```

Textarea.watch()

JavaScript 1.2+
Nav4+

Syntax

```
textarea.watch(property, function)
```

Description

The `watch()` method of the `Textarea` object is used to turn on the watch for a particular *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 8.510 shows how the `watch()` method is used to start watching the value property.

Listing 8.510 Example of the `watch()` Method

```

<html>
<head>
  <script type="text/javascript" language="JavaScript1.2">

    function alertme(id, oldValue, newValue){
      document.writeln("ID ( " + id + " ) changed from " + oldValue + " to ");
      document.writeln(newValue + "<br>");
      return newValue;
    }

    function check(){
      document.myForm.myItem.watch("value", alertme);
      document.myForm.myItem.value="Send";
    }
  </script>
</head>

```

Listing 8.510 Continued

```
<body onload="check()">
<form name="myForm">
  <textarea name="myItem" rows="3" cols="30" value="Submit"></textarea>
</form>
</body>
</html>
```

untaint()

JavaScript1.1
Nav3

Syntax

```
untaint(property)
untaint(variable)
untaint(function)
untaint(object)
```

Description

The `untaint()` method, a security measure that was only implemented in JavaScript 1.1, allowed a developer to allow return values to be used by and propagated to other scripts. This method does not change the data element passed to it, but rather returns an unmarked reference to the element.

Because `untaint()` and the functionality of data tainting was removed in JavaScript 1.2, you should avoid using this method. You should use it only if you have a specific security reason for compatibility with Navigator 3 browsers. See Chapter 1 for more information on the security model that is now used in Navigator browsers.

Example

Listing 8.511 simply untaints a variable that is defined in a separate window.

Listing 8.511 Use of the `untaint()` Method, Which Is No Longer Supported

```
<script language="JavaScript1.1" type="text/javascript">
<!--

// Store the untainted variable from the second window
// in the variable utMyVar.
var utMyvar = untaint(myWin.myVar);

// -->
</script>
```

Window

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Core client-side JavaScript object.

Description

The Window object is one of the top-level JavaScript objects that are created when a <body>, <frameset>, or <frame> tag is encountered. Instances of this object can also be created by using the Window.open() method. Table 8.39 lists the properties and methods associated with the Window object.

Table 8.39 *Methods and Properties of the Window Object*

Type	Item	Description
Method	alert()	Displays an alert dialog box with the text string passed.
	atob()	Decodes a string that has been encoded using base-64 encoding.
	back()	Loads the previous page in place of the window instance.
	blur()	Removes the focus from a window.
	btoa()	Encodes a string using base-64 encoding.
	captureEvents()	Sets the window to capture all events of a specified type.
	clearInterval()	Clears the interval set with the setInterval method.
	clearTimeout()	Clears the timeout set with the setTimeout method.
	close()	Closes the instance of the window.
	confirm()	Displays a confirmation dialog box.
	crypto.random()	Returns a random string that's length is specified by the number of bytes passed to the method.
	crypto.signText()	Returns a string of encoded data that represents a signed object.
	disableExternalCapture()	Disables external event capturing.
	enableExternalCapture()	Enables external event capturing for the pages loaded from other servers.
	find()	Displays a Find dialog box where the user can enter text to search the current page.

Table 8.39 Continued

Type	Item	Description
	<code>focus()</code>	Assigns the focus to the specified window instance.
	<code>forward()</code>	Loads the next page in place of the window instance.
	<code>handleEvent()</code>	Invokes the handler for the event passed.
	<code>home()</code>	Loads the user's specified home page in place of the window instance.
	<code>moveBy()</code>	Moves the window by the amounts specified.
	<code>moveTo()</code>	Moves the window to the specified location.
	<code>open()</code>	Opens a new instance of a window.
	<code>print()</code>	Invokes the Print dialog box so that the user can print the current window.
	<code>prompt()</code>	Displays a prompt dialog box.
	<code>releaseEvents()</code>	Releases the captured events of a specified type.
	<code>resizeBy()</code>	Resizes the window by the specified amount.
	<code>resizeTo()</code>	Resizes the window to the specified size.
	<code>routeEvent()</code>	Passes the events of a specified type to be handled natively.
	<code>scroll()</code>	Scrolls the document in the window to a specified location.
	<code>scrollBy()</code>	Scrolls the document in the window by a specified amount.
	<code>scrollTo()</code>	Scrolls the document, both width and height, in the window to a specified location.
	<code>setHotKeys()</code>	Enables or disables hot keys in a window that does not have menus.
	<code>setInterval()</code>	Invokes a function or evaluates an expression every time the number of milliseconds has passed.
	<code>setResizable()</code>	Specifies whether a user can resize a window.
	<code>setTimeout()</code>	Invokes a function or evaluates an expression when the number of milliseconds has passed.
	<code>setZOptions()</code>	Specifies the z-order stacking behavior of the window.

Type	Item	Description
Property	<code>stop()</code>	Stops the current window from loading other items within it.
	<code>unwatch()</code>	Used to turn off the watch for a particular property.
	<code>watch()</code>	Used to turn on the watch for a particular property.
	<code>closed</code>	Specifies whether the window instance has been closed.
	<code>crypto</code>	Actually a subobject of the <code>window</code> object, which allows access to the browser's encryption features.
	<code>defaultStatus</code>	Is the default message in the window's status bar.
	<code>document</code>	References all the information about the document within this window. See the <code>Document</code> object for more information.
	<code>frames</code>	References all the information about the frames within this window. See the <code>Frame</code> object for more information.
	<code>history</code>	References the URLs that the user has visited.
	<code>innerHeight</code>	Contains the height of the display area of the current window in pixels.
	<code>innerWidth</code>	Contains the width of the display area of the current window in pixels.
	<code>length</code>	Represents the number of frames in the current window.
	<code>location</code>	Contains the current URL loaded into the window.
	<code>locationbar</code>	Reference to the browser's location bar.
	<code>menubar</code>	Reference to the browser's menu bar.
	<code>name</code>	Name of the window.
	<code>offScreenBuffering</code>	Specifies whether updates to a window are performed in an offscreen buffer.
	<code>opener</code>	Contains the name of the window from which a second window was opened.
	<code>outerHeight</code>	Contains the height of the outer area of the current window in pixels.
	<code>outerWidth</code>	Contains the width of the outer area of the current window in pixels.

Table 8.39 Continued

Type	Item	Description
	pageXOffset	Contains the x coordinate of the current window.
	pageYOffset	Contains the y coordinate of the current window.
	parent	Reference to the uppermost window that is displaying the current frame.
	personalbar	Reference to the browser's personal bar.
	screenX	Specifies the x coordinate of the left edge of the window.
	screenY	Specifies the y coordinate of the top edge of the window.
	scrollbars	Reference to the browser's scrollbars.
	self	Reference to the current window.
	status	Reference to the message in the window's status bar.
	statusbar	Reference to the browser's status bar.
	toolbar	Reference to the browser's toolbar.
	top	Reference to the uppermost window that is displaying the current frame.
	window	Reference to the current window.

Any of the methods or properties of this object can have the instance name left off if it refers to the window in which they are invoked. Setting the `status` property, for instance, would set the status in the current window, whereas setting `myWin.status` sets the status in the window named `myWin`. The only exception to this rule is in using the `location` property and the `close()` and `open()` methods, which must take the instance's name when called within an event handler.

NOTE

Each of the entries for the `Window` object's methods and properties in this chapter will be preceded by the `Window` object name in its syntax definition.

Example

Listing 8.512 contains a button. When the button is clicked, a new window instance is created using the `open()` method. Information is then written to the window before the stream is closed. There is also a button in the new window to close it.

Listing 8.512 Creating a New Instance of the Window Object

```

<html>
<head>

<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin() function.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ' ,titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizable=no';
    var myFeatures = myBars + ' , ' + myOptions;
    var myReadme = "Welcome to Pure JavaScript!\n" +
        "-----\n" +
        'You can enter some text here.'

    // Open the window. Give the window instance the name newDoc and
    // name the document in the window myWin.
    var newWin = open('', 'myWin', myFeatures);

    newWin.document.writeln('<form>');
    newWin.document.writeln('<table cellpadding="0" cellspacing="0"
border="1">');
    newWin.document.writeln('<tr valign="top" bgcolor="#000099"><td>');
    newWin.document.writeln('<font size="-1" color="ffffff"><b>');
    newWin.document.writeln('&nbsp;&nbsp;&nbsp;Readme</b></font>');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('<tr valign="top"><td>');
    newWin.document.writeln('<textarea cols="45" rows="7" wrap="soft">');
    newWin.document.writeln(myReadme + '</textarea>');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('<tr valign="bottom" align="right">');
    newWin.document.writeln('<td bgcolor="#c0c0c0"><td>');
    newWin.document.writeln('<input type="button" value="Close">');
    newWin.document.writeln('< onclick="window.close()">');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('</table></form>');

```

Listing 8.512 Continued

```
// Close the stream to the document and bring the window to the front.
newWin.document.close();
newWin.focus();
}

// -->
</script>
</head>
<body>
<form>
  <b>Click the following button to open a new window: </b>
  <input type="button" value="Open" onclick='openWin()'>
</form>
</body>
</html> >
```

window.alert()

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

window.alert(string)

Description

The `alert()` method of the Window object displays an alert dialog box when invoked. The value of the string passed to the method is displayed in the box.

Example

Listing 8.513 pops up an alert box when the script is loaded. The result of running this script can be seen in Figure 8.4.

Listing 8.513 An Alert Box

```
<script language="JavaScript" type="text/javascript">
<!--

alert('Here is an alert dialog box.');
```

// -->
</script>

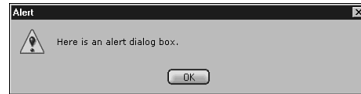


Figure 8.4

An alert box created with the `window.alert()` method.

window.atob()

JavaScript1.2
Nav4

Syntax

`window.atob(data)`

Description

The `atob()` method of the `Window` object, which was only implemented in Navigator 4, is used to decode *data* that was encoded in base-64 encoding.

Example

Listing 8.514 shows how you can encode data using the `btoa()` method, and then decode it back using `atob()`.

Listing 8.514 Using the `atob()` Method to Decode Data

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
      var myString = new String();
      myString = "Hello world";

      // Write out encoded and decoded versions of the string.
      document.write(window.btoa(myString));
      document.write(window.atob(myString));
    <!-->
  </script>
</body>
</html>
```

window.back()

JavaScript1.2+, JScript3.0
Nav4+, IE4+

Syntax

window.back()

Description

The `back()` method of the `Window` object simulates the user clicking the Back button on the browser. It returns the browser's page or frame to the previous page or frame.

Example

Listing 8.515 has two buttons. One of the buttons takes the browser back one page and the other button takes it forward. Note that there has to be a back and forward page during your session for the button to have somewhere to go.

Listing 8.515 Using the back() Method

```
<html>
<body>
<form>
  <input type="button" value="Back" onclick="window.back()">
  <input type="button" value="Forward" onclick="window.forward()">
</form>
</body>
</html>
```

window.blur()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

window.blur()

Description

The `blur()` method of the `Window` object removes the focus from the window. Be careful when using this method in conjunction with the `focus()` method of objects. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.516 has two buttons. When the user clicks the Open button, a second, smaller window is opened. If the Blur button is clicked, the focus is removed from the parent window and placed on the child window.

Listing 8.516 Using the `blur()` Method to Remove the Focus from a Window

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin() function
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += 'titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizable=no';
    var myFeatures = myBars + ',' + myOptions;

    // Open a child window.
    newWin = open('', 'myDoc', myFeatures);

    newWin.document.writeln('Here is the child window');

    // Close the stream to the document.
    newWin.document.close();

    // Return focus to the parent window.
    self.focus();
}

// Define the blurWin() function.
function blurWin(){

    // Blur the parent window and focus on the child.
    self.blur();
    newWin.focus();
}

// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Open" onclick='openWin()'>
    <input type="button" value="Blur" onclick='blurWin()'>
</form>
</body>
</html>

```

window.btoa()

JavaScript1.2
Nav4

Syntax

window.btoa(data)

Description

The `btoa()` method of the `Window` object, which was only implemented in Navigator 4, is used to encode *data* in base-64 encoding.

Example

Listing 8.517 shows how you can encode data using the `btoa()` method, and then decode it an back using `atob()`.

Listing 8.517 Using the btoa() Method to Decode Data

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
      var myString = new String();
      myString = "Hello world";

      // Write out encoded and decoded versions of string
      document.write(window.btoa(myString));
      document.write(window.atob(myString))
    <!-->
  </script>
</body>
</html>
```

window.captureEvents()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.captureEvents(event)
window.captureEvents(event1 | event2 | eventN)

Description

The `captureEvents()` method of the `Window` object captures all the events of the event type passed. Because you can capture the events that are natively handled by the language itself, programmers can now define a function to handle events in a manner they want. If you have multiple events that you want to capture, separate them with the pipe (`|`) character. The types of events that can be captured are as follows:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`
- `Event.MOUSEOVER`
- `Event.MOUSEUP`
- `Event.MOVE`
- `Event.RESET`
- `Event.RESIZE`
- `Event.SELECT`
- `Event.SUBMIT`
- `Event.UNLOAD`

After an event has been captured, you can define a function to replace the built-in method for handling the event.

Example

Listing 8.518 has a single text box. The script in the `<head>` of the document specifies a function to handle all `onClick` events in the window. To be able to do this, the `captureEvents()` method had to be used to capture all events of type `Event.CLICK`. When the page itself is clicked, a counter, which is displayed in the text box, is incremented.

Listing 8.518 Capturing Events with the `Window.captureEvents()` Method

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text box.
    window.document.myForm.myText.handleEvent(e);

}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText" onclick='changeText()'>
</form>
</body>
</html>
```

`window.clearInterval()`

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

`window.clearInterval(interval)`

Description

The `clearInterval()` method of the Window object clears the interval that is passed to the method. The interval that is passed has to be previously defined using the `setInterval()` method.

Example

Listing 8.519 sets an interval in the `<head>` of the document that displays a counter in a text box on the page. An interval is set to update the counter in the text box every five seconds. There is also a button on the page that can be clicked to clear the interval and stop the counting.

Listing 8.519 Clearing an Interval with the `clearInterval()` Method

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Create a variable to hold a counter.
var counter = 1;

// Define a function to display the counter.
function startCounter(){
    document.myForm.myText.value = counter++;
}

// Define a function to stop the counter.
function stopCounter(){
    window.clearInterval(myInterval);
}

// Set the interval to call the function every 5 seconds.
var myInterval = window.setInterval("startCounter()", 5000)

// -->
</script>
</head>
<body onload="startCounter()">
<form name="myForm">
    <input type="text" size="2" value="" name="myText">
    <input type="button" value="Clear Interval" onclick="stopCounter()">
</form>
</body>
</html>
```

window.clearTimeout()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
window.clearTimeout(timeout)
```

Description

The `clearTimeout()` method of the Window object clears the timeout passed to the method. The timeout that is passed has to be previously defined using the `setTimeout()` method.

Example

Listing 8.520 has a button and text box. By the default, the time will be displayed in the text box after five seconds. This is done using the `setTimeout()` method. If the button is clicked, a function is called that invokes the `clearTimeout()` method preventing the time from being displayed in the text box.

Listing 8.520 Using the `clearTimeout()` Method

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to show the time.
function showTime(){
    myTime = new Date();
    myTime = myTime.getHours() + ":" + myTime.getMinutes() + ":";
    myTime += myTime.getSeconds();
    document.myForm.myText.value = myTime;
}

// Define a function to stop the display of the time.
function stopTime(){
    window.clearTimeout(myTimeout);
}

// Set the interval to call the function after 5 seconds.
var myTimeout = window.setTimeout("showTime()", 5000)

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText">
```



```

    <input type="button" value="Clear Timeout" onclick="stopTime()">
  </form>
</body>
</html>

```

window.close()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
window.close()
```

Description

The `close()` method of the Window object is used to close browser windows. Even though this method was first introduced in JavaScript 1.0, there have been some changes. In the first version, this method could be used to close any window. In JavaScript 1.1, it was restricted to close only windows opened using JavaScript. In JavaScript 1.2, you must have the `UniversalBrowserWrite` privilege to unconditionally close a window.

Example

Listing 8.521 has a button that opens a window. Within the opened window there is a Close button. Clicking this button invokes the `close()` method and closes the browser window.

Listing 8.521 Using the close() Method to Close a Window

```

<html>
<head>

<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin() function.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ',' + myOptions;
    var myReadme = "Welcome to Pure JavaScript!\n" +
        ".....\n" +
        "Click the Close button to invoke the close() " +
        "method and close the window."

```

Listing 8.521 Continued

```
// Open the window. Give the window instance the name newDoc and
// name the document in the window myDoc.
var newWin = open('', 'myDoc', myFeatures);

newWin.document.writeln('<form>');
newWin.document.writeln('<table cellpadding="0" cellspacing="0"
border="1">');
newWin.document.writeln('<tr valign="top" bgcolor="#000099"><td>');
newWin.document.writeln('<font size="-1" color="ffffff"><b>');
newWin.document.writeln('&nbsp;&nbsp;&nbsp;Readme</b></font>');
newWin.document.writeln('</td></tr>');
newWin.document.writeln('<tr valign="top"><td>');
newWin.document.writeln('<textarea cols="45" rows="7" wrap="soft">');
newWin.dpcument.writeln(myReadme + '</textarea>');
newWin.document.writeln('</td></tr>');
newWin.document.writeln('<tr valign="bottom" align="right">');
newWin.document.writeln(' bgcolor="#c0c0c0"><td>');

// Write the close() method on the new window. Invoke it with an onClick
// event.
newWin.document.writeln('<input type="button" value="Close">');
newWin.document.writeln(' onclick="window.close()">');

newWin.document.writeln('</td></tr>');
newWin.document.writeln('</table></form>');

// Close the stream to the document and bring the window to the front.
newWin.document.close();
newWin.focus();
}

// -->
</script>
</head>
<body>
<form>
  <b>Click the following button to open a new window: </b>
  <input type="button" value="Open" onclick='openWin()'>
</form>
</body>
</html>
```

window.closed

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

window.closed

Description

The `closed` property of the `Window` object returns a Boolean value specifying whether the window instance it is referencing is closed. If the window is still open, the property returns `false`. If it is closed, the property returns `true`.

Example

Listing 8.522 has two buttons. When the Open button is clicked, a second window is opened and focused. When the Check button is clicked, the script checks to see whether the window is still open. If it is, the text in the text area of the second window is changed. If it is not, an alert dialog box is displayed.

Listing 8.522 *Using the `closed` Property to See Whether a Window Is Still Open*

```
<html>
<head>

<script language="JavaScript1.1" type="text/javascript">
<!--

// Define the openWin() function.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ',' + myOptions;
    var myReadme = "Welcome to Pure JavaScript!\n" +
        "-----\n" +
        "Click the Close button to invoke the close() " +
        "method and close the window."

    // Open the window. Give the window instance the name newWin and
    // name the document in the window myDoc.
    newWin = open('', 'myDoc', myFeatures);

    newWin.document.writeln('<form name="secondForm">');
    newWin.document.writeln('<table cellpadding="0" cellspacing="0" ' +
border="1">');
    newWin.document.writeln('<tr valign="top" bgcolor="#000099"><td>');
    newWin.document.writeln('<font size="-.1" color="#ffffff"><b>');
    newWin.document.writeln('&nbsp;&nbsp;&nbsp;Readme</b></font>');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('<tr valign="top"><td>');
    newWin.document.writeln('<textarea name="myTextArea" cols="45" rows="7">');
    newWin.document.writeln(' wrap="soft"> ' + myReadme + '</textarea>');
    newWin.document.writeln('</td></tr>');
```

Listing 8.522 Continued

```
newWin.document.writeln('<tr valign="bottom" align="right">');
newWin.document.writeln(' bgcolor="#c0c0c0"><td>');
newWin.document.writeln('<input type="button" value="Close" ');
newWin.document.writeln('onclick="window.close()">');
newWin.document.writeln('</td></tr>');
newWin.document.writeln('</table></form>');

// Close the stream to the document and bring the window to the front.
newWin.document.close();
newWin.focus();
}

function checkWin(){

    // Use the closed property to see if the window has been closed.
    if(newWin.closed){
        alert("Sorry, the window has been closed.");
    }else{
        var myText = "This window is still opened";
        newWin.document.secondForm.myTextArea.value = myText;
        newWin.focus();
    }
}
// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Open" onclick='openWin()'>
    <input type="button" value="Check" onclick='checkWin()'>
</form>
</body>
</html>
```

window.confirm()

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

`window.confirm(string)`

Description

The `confirm()` method of the `Window` object displays a confirmation dialog box when invoked. The value of the string passed to the method is displayed in the box. This box will contain both an OK and a Cancel button. The method returns a Boolean value of `true` if the user clicks OK and `false` if the user clicks Cancel.

Example

Listing 8.523 pops up a confirmation box when the script is loaded to see whether the user wishes to proceed. After the user makes a decision, the script writes his choice to the page. The result of running this script can be seen in Figure 8.5.

Listing 8.523 A Confirm Box Using the `confirm()` Method

```
<script language="JavaScript" type="text/javascript">
<!--

// Ask the user if they want to proceed.
if(confirm("Are you sure you want to do this?")){
    document.write("You clicked the OK button");
}else{
    document.write("You clicked the Cancel button");
}

// Close the stream to the document.
document.close();

// -->
</script>
```

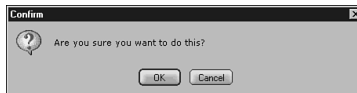


Figure 8.5

A confirmation box created with the `Window.confirm()` method.

window.crypto

JavaScript1.2
Nav4

Syntax

window.crypto.method

Description

The `crypto` property of the `Window` object is used to access the encryption functions available within Navigator 4.

Example

Listing 8.524 shows how you can use the `crypto` property to access the `random()` method within Navigator's encryption features.

Listing 8.524 Using the `crypto` Property to Access Encryption Methods

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
    // Writes out the encryption random number.
    document.write(crypto.random(16));
    //-->
  </script>
</body>
</html>
```

window.crypto.random()

JavaScript1.2
Nav4

Syntax

`window.crypto.random(num)`

Description

The `random` method of the `window.crypto` object is used to create a random number of length *num* in bytes. This method is part of the encryption functions available within the Navigator 4.

Example

Listing 8.525 shows how you can use the `crypto` property to access the `random()` method within Navigator's encryption features.

Listing 8.525 Using the `random()` Method to Generate a Random Number

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
    // Writes out the encryption random number.
    document.write(crypto.random(16));
    //-->
  </script>
</body>
</html>
```

window.crypto.signText()

JavaScript1.2
Nav4

Syntax

```
window.crypto.signText(text, style)
```

```
window.crypto.signText(text, style, authority1, ..., authorityN)
```

Description

The `signText()` method of the `window.crypto` object is used to sign *text*. Depending on the *style* of signing, which can be `ask` or `auto`, the browser might pop up a dialog with a list of possible certificates to sign the *text* with. You can optionally pass 1 or more *authority* arguments that will allow you to pass Certificate Authorities.

After the data has been encoded and sent to the server, the server will decode and verify it. If a failure occurs, one of the following codes will be returned:

- `error: noMatchingCert`—Specifies that the user's certificate does not match one of the passed authority arguments.
- `error: userCancel`—Specifies that the user cancelled the ask dialog box without submitting a certificate.
- `error: internalError`—Specifies that an internal error has occurred while processing the data.

Example

Listing 8.526 prompts the user for some text, and then prompts him before signing the text. After being signed, the text could then be passed to the server.

Listing 8.526 Using the `signText()` Method

```
<html>
<script type="text/javascript" language="JavaScript1.2">
<!--

    // Prompt for text.
    myText = window.prompt("Please enter some text", "");

    // Sign the text.
    crypto.signText(myText, ask);
-->
</script>
</html>
```

window.defaultStatus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
window.defaultStatus = string
```

Description

The `defaultStatus` property of the `Window` object reflects the message that is displayed in the status bar of the browser. Note that in JavaScript 1.1, this property was tainted. See Chapter 1 for more information on JavaScript security and data tainting.

Example

Listing 8.527 shows how you can set the default status to be displayed after a document has finished loading. This is done in conjunction with the `onLoad` event handler within the `<body>` tag.

Listing 8.527 Setting the Default Status of a Page

```
<body onload="window.defaultStatus='Please make a selection'">
```

window.disableExternalCapture()

JavaScript1.2+
Nav4+

Syntax

```
window.disableExternalCapture()
```

Description

The `disableExternalCapture()` method of the `Window` object disables any external event capturing set up using the `enableExternalCapture()` method. The functionality of this method provides the capturing of events in frames loaded from a different server. Before you can enable the capturing of these external events, you must first obtain `UniversalBrowserWrite` privileges. After they have been obtained and the method has been invoked, use the `Window.captureEvents()` method to specify the events you want to capture.

NOTE

For more information on privileges, JavaScript security, and signed scripts, see Chapter 1.

Example

Listing 8.528 enables external event capturing when the document loads. The document itself has a button that, when clicked, calls a function to disable the external event captures.

Listing 8.528 Disabling External Event Capturing

```
<html>  
<head>  
<script language="JavaScript1.2" type="text/javascript">  
<!--
```



```
// Ask the user for permission to enable the UniversalBrowserWrite
// privilege.
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");

// Enable the external capturing of events.
window.enableExternalCapture();

// Specifically capture submit events.
window.captureEvents(Event.SUBMIT);

// Define a function to turn off these external event captures.
function turnOffEvents(){
    window.disableExternalCapture();
    alert("You have sucessfully turned off external event captures");
}
// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Disable External Capturing"
        onclick="turnOffEvents()">
</form>
</body>
</html>
```

window.document

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

Creates Instance of Document Object.

```
window.document.event
window.document.method()
window.document.property
```

Description

The document property, which is a child object of the Window object, is a core JavaScript object that is created when instances of the <body> tag are encountered. The properties, methods, and events associated with this object are in Table 8.40.

Table 8.40 Event Handlers, Methods, and Properties Used by the Window.document Property

Type	Item	Description
Event Handler	onClick	Executes code when the document is clicked.
	ondblclick	Executes code when the document is double-clicked.
	onkeydown	Executes code when a key is pressed down. This occurs before an onKeyPress event handler and was added in JavaScript 1.2.
	onkeypress	Executes code when a key is pressed down immediately after an onKeyDown event handler. This event handler was added in JavaScript 1.2.
	onkeyup	Executes code when a key is released. This was added in JavaScript 1.2.
	onmousedown	Executes code when the mouse button is pressed down.
	onmouseup	Executes code when the mouse button is released.
Method	captureEvents()	Allows you to capture all events of the type passed in the document. Note that this method was added in JavaScript 1.2.
	close()	Closes the stream to the document.
	getSelection()	Returns the currently selected text. Note that this method was added in JavaScript 1.2.
	handleEvent()	Invokes the handler for the event specified and was added in JavaScript 1.2.
	open()	Opens a stream to the document.
	releaseEvents()	Releases the events that you have captured of the type passed in the document. Note that this method was added in JavaScript 1.2.
	routeEvent()	Passes the specified event along the normal route of execution. Note that this method was added in JavaScript 1.2.
	write() writeln()	Writes the string passed to the document. Writes the string, followed by a newline character, to the document.
Property	alinkColor	Specifies the alink attribute of the <body> tag.
	anchors	Array containing each <a> tag in a document.
	applets	Array containing each <applet> tag in a document. Note that this property was added in JavaScript 1.1.
	bgColor	Specifies the bgColor attribute of the <body> tag.

Type	Item	Description
	cookie	Specifies a cookie.
	domain	Specifies the domain that served the document. Note that this property was added in JavaScript 1.1.
	embeds	Array containing each <code><embed></code> tag in a document. Note that this property was added in JavaScript 1.1.
	fgColor	Specifies the <code>text</code> attribute of the <code><body></code> tag.
	formName	The actual name of each <code><form></code> in a document. Note that this property was added in JavaScript 1.1.
	forms	Array containing each <code><form></code> tag in a document. Note that this property was added in JavaScript 1.1.
	images	Array containing each <code></code> tag in a document. Note that this property was added in JavaScript 1.1.
	lastModified	Specifies the date the document was last changed.
	layers	Array containing each <code><layer></code> tag in a document. Note that this property was added in JavaScript 1.2.
	linkColor	Specifies the <code>link</code> attribute of the <code><body></code> tag.
	links	Array containing each <code><a></code> and <code><area></code> tag in a document.
	plugins	Array containing each plug-in in a document. Note that this property was added in JavaScript 1.1.
	referrer	Specifies the referral URL.
	title	Contains the text between the beginning <code><title></code> and ending <code></title></code> tags.
	URL	Specifies the URL of the document.
	vlinkColor	Specifies the <code>vlink</code> attribute of the <code><body></code> tag.

See the entries in this chapter for the Document object for more information on each of these events, methods, and properties.

Example

Listing 8.529 uses the `write()` method of the document property to write text to the user's page.

Listing 8.529 Accessing Methods of the document Property

```
<script language="JavaScript" type="text/javascript">
<!--

// Simply writes out the Hello, World! text
window.document.write("Hello, World!");

// -->
</script>
```

window.enableExternalCapture()

JavaScript1.2+
Nav4+

Syntax

window.enableExternalCapture(event)

Description

The `enableExternalCapture()` method of the `Window` object enables external event capturing of the event that is passed to the method. This method provides the capturing of events in frames loaded from a different server. Before you can enable the capturing of these external events, you must first obtain `UniversalBrowserWrite` privilege. Obtaining this privilege will send a security dialog box to the user to decide whether to accept the request.

After it has been obtained and the method has been invoked, you use the `Window.captureEvents()` method to specify the events you want to capture. To remove the ability to capture these events, you can invoke the `Window.disableExternalCapture()` method. The types of events that can be captured are as follows:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`

- Event.MOUSEOVER
- Event.MOUSEUP
- Event.MOVE
- Event.RESET
- Event.RESIZE
- Event.SELECT
- Event.SUBMIT
- Event.UNLOAD

NOTE

For more information on privileges, JavaScript security, and signed scripts, see Chapter 1.

Example

Listing 8.530 enables external event capturing when the document loads.

Listing 8.530 Enabling External Event Capturing

```
<script language="JavaScript1.2" type="text/javascript">
<!--

// Ask the user for permission to enable the UniversalBrowserWrite
// privilege.
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");

// Enable the external capturing of events.
window.enableExternalCapture();

// Specifically capture submit events.
window.captureEvents(Event.SUBMIT);

// -->
</script>
```

window.find()

JavaScript1.2, JScript3.0
Nav4, IE4

Syntax

```
window.find()
```

Description

The `find()` method of the `Window` object displays a find dialog box when invoked. This allows a user to search for a string in the page from which it was invoked.



NOTE

This method was only supported in the version 4 browsers, and has been removed since then.

Example

Listing 8.531 has a function that pops up a Find box when it is called.

Listing 8.531 A Find Box That Can Be Used to Search for Text in the Document

```
<script language="JavaScript1.2" type="text/javascript">
<!--

// Displays the Find window.
function mySearch(){
    window.find();
}

// -->
</script>
```

window.focus()

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

`window.focus()`

Description

The `focus()` method of the `Window` object places focus on the window. Be careful when using this method in conjunction with the `blur()` method of objects. It can lead to a focus/blur loop, where the browser blurs a focus as soon as it is done, and vice versa.

Example

Listing 8.532 has a button. When the user clicks the Open button, a second, smaller window is opened, and the focus is placed back on the parent window.

Listing 8.532 Using the `focus()` Method to Remove Focus from a Window

```
<html>
<head>
</head>
```

```

<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin() function.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizable=no';
    var myFeatures = myBars + ', ' + myOptions;

    // Open a child window.
    newWin = open('', 'myDoc', myFeatures);

    newWin.document.writeln('Here is the child window');

    // Close the stream to the document.
    newWin.document.close();

    // Return focus to the parent window.
    self.focus();
}

// -->
</script>
<body>
<form>
    <input type="button" value="Open" onclick='openWin()'>
</form>
</body>
</html>

```

window.forward()

JavaScript1.2+, Jscript3.0
Nav4+, IE4+

Syntax

window.forward()

Description

The `forward()` method of the `Window` object simulates the user clicking the Forward button on the browser. It returns the browser's page or frame to the next page or frame in its history.

Example

Listing 8.533 has two buttons. One of the buttons takes the browser back one page and the other button takes it forward. Note that there has to be a back and forward page during your session for the button to have somewhere to go.

Listing 8.533 *Using the forward() Method to Take the User to the Next Page in His History*

```
<html>
<body>
<form>
  <input type="button" value="Back" onclick="window.back()">
  <input type="button" value="Forward" onclick="window.forward()">
</form>
</body>
</html>
```

window.frames

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
window.frames["frameName"]
window.frames[num]
```

Description

The frames property of the Window object contains an array that stores each frame instance, created with the <frame> tag, in a document. Array entries of the child frame can be referenced either by index number or by the name assigned by the name attribute of the <frame> tag.

Example

Listing 8.534 uses the length property of frames array and a for loop to access the name of each frame in the window. This information is then written to the document window.

Listing 8.534 *Example of Using the frames Property*

```
<script language="JavaScript" type="text/javascript">
<!--

// Use a for loop to write out the name of each frame.
for(var i = 0; i <= window.frames.length; i++){
  newWin.document.write("The name of frame #" + i);
  newWin.document.write(" is " + window.frames[i].name + "<br>");
}
// -->
</script>
```

window.frames.length

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
window.frames["frameName"].length  
window.frames[num].length
```

Description

The `length` sub property of the `frames` property of the `Window` object contains the number of frame instances in a document created with the `<frame>` tag.

Example

Listing 8.535 uses the `length` property of the `frames` array and a `for` loop to access the name of each frame in the window. This information is then written to the document window.

Listing 8.535 Using the length Property

```
<script language="JavaScript" type="text/javascript">  
<!--  
  
// Use a for loop to write out the name of each frame.  
for(var i = 0; i <= window.frames.length; i++){  
    newWin.document.write("The name of frame #" + i);  
    newWin.document.write(" is " + window.frames[i].name + "<br>");  
}  
// -->  
</script>
```

window.handleEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
window.object.handleEvent(event)
```

Description

The `handleEvent()` method of the `Window` object invokes the handler for the event specified of the specified object. This method was added in JavaScript 1.2.

Example

Listing 8.536 has a single text box. The script tells the browser that it wants to intercept all `Click` events and that it wants the `myClickHandler` function to handle them. Within this function, the `handleEvent()` method of the text box has been specified to handle the click.

When the user clicks anywhere on the page, the `onClick` event handler in the `<input type="text">` tag calls a function to change the text in the text box. The change is nothing more than a simple number that is incremented, counting the number of times the page has been clicked.

Listing 8.536 *Using the `handleEvent()` Method of a Window Object to Handle All Clicks on a Page*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK)
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text box.
    window.document.myForm.myText.handleEvent(e);
}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}
// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText" onclick='changeText()'>
</form>
</body>
</html>
```

window.history

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

```

window.history[num]
window.history.method()
window.history.property

```

Description

The history property of the Window object is actually one of the core JavaScript objects. This object contains an array of the names and URLs of the pages the window has visited. A specific URL in the history array can be accessed by specifying the indexed location, *num*, that represents the URL about which you want to retrieve information.

Also, as defined in the syntax definition, the methods and properties of this object are also accessible for programming use. Table 8.41 has a list of each of these, followed by a description.

Table 8.41 *Methods and Properties Used by the Window.history Property*

Type	Item	Description
Method	back	References the URL that is located one page back from the current page.
	forward	References the URL that is located one page ahead of the current page.
	go	Loads the URL passed to the method. This can be in relation to the current URL or a string representing part or the whole URL you want to access.
Property	current	Reflects the current URL of the window. This property was added in JavaScript 1.1.
	length	Reflects the number of URLs in the history of the window.
	next	Reflects the next URL in the history in relation to the current URL. This property was added in JavaScript 1.1.
	previous	Reflects the last URL in the history in relation to the current URL. This property was added in JavaScript 1.1.

For more information on the History object and its properties and methods, see its entry in this chapter.

Example

Listing 8.537 has two buttons that allow a user to move forward and back in his history.

Listing 8.537 Using the history Array to Access Pages Visited

```
<html>
<body>
<form>
  <input type="button" value="Back" onclick="window.history.back()">
  <input type="button" value="Forward" onclick="window.history.forward()">
</form>
</body>
</html>
```

window.home()

JavaScript1.2+, JScript3.0
Nav4+, IE4+

Syntax

`window.home()`

Description

The `home()` method of the `Window` object simulates the user clicking the Home button on the browser. It takes the browser to the user's specified home page.

Example

Listing 8.538 has a single button that, when clicked, takes the browser to the user's home page.

Listing 8.538 Using the home() Method to Go to the User's Home Page

```
<form>
<h3>Home James!</h3>
  <input type="button" value="Home" onclick="window.home()">
</form>
```

window.innerHeight

JavaScript1.2+
Nav4+

Syntax

`window.innerHeight`

Description

The `innerHeight` property of the `Window` object references the pixel height of the document within the browser's frame. This does not include any of the toolbars or other "chrome" that makes up the frame itself.

Example

Listing 8.539 has a button that, when clicked, opens up a second, smaller window. The `innerHeight` property is written to this new window.

Listing 8.539 *Using the `innerHeight` Property*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to open a small window.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ' ,titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ' , ' + myOptions;

    // Open the window. Give the window instance the name newDoc and
    // name the document in the window myDoc.
    var newWin = open(' ', 'myDoc', myFeatures);

    // Write the window height and width properties to the new window.
    newWin.document.writeln('<h4>Properties for this Window</h4>');
    newWin.document.writeln('innerHeight: ' + newWin.innerHeight + '<br>');
    newWin.document.writeln('innerWidth: ' + newWin.innerWidth + '<br>');
    newWin.document.writeln('outerHeight: ' + newWin.outerHeight + '<br>');
    newWin.document.writeln('outerWidth: ' + newWin.outerWidth + '<br>');
    newWin.document.writeln('<form>');
    newWin.document.writeln('<input type="button" value="Close">');
    newWin.document.writeln(' onclick="window.close()">');
    newWin.document.writeln('</form>');

    // Close the stream to the document.
    newWin.document.close();

}
// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Open" onclick="openWin()">
</form>
</body>
</html>
```

window.innerWidth

JavaScript1.2+
Nav4+

Syntax

`window.innerWidth`

Description

The `innerWidth` property of the `Window` object references the pixel width of the document within the browser's frame. This does not include any of the toolbars or other "chrome" that makes up the frame itself.

Example

Listing 8.540 has a button that, when clicked, opens up a second, smaller window. The `innerWidth` property is written to this new window.

Listing 8.540 Using the `innerWidth` Property

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to open a small window.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ',' + myOptions;

    // Open the window. Give the window instance the name newWin and
    // name the document in the window myDoc.
    var newWin = open('', 'myDoc', myFeatures);

    // Write the window height and width properties to the new window.
    newWin.document.writeln('<h4>Properties for this Window</h4>');
    newWin.document.writeln('innerHeight: ' + newWin.innerHeight + '<br>');
    newWin.document.writeln('innerWidth: ' + newWin.innerWidth + '<br>');
    newWin.document.writeln('outerHeight: ' + newWin.outerHeight + '<br>');
    newWin.document.writeln('outerWidth: ' + newWin.outerWidth + '<br>');
    newWin.document.writeln('<form>');
    newWin.document.writeln('<input type="button" value="Close">');
    newWin.document.writeln(' onclick="window.close()">');
    newWin.document.writeln('</form>');
```

```
// Close the stream to the document.
newWin.document.close();

}
// -->
</script>
</head>
<body>
<form>
  <input type="button" value="Open" onclick="openWin()">
</form>
</body>
</html>
```

window.length

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`window.length`

Description

The `length` property of the `Window` object represents the number of frames within a window. This returns the same results as `Window.frames.length`.

Example

Listing 8.541 shows a function that can be used to return the number of frames in a window.

Listing 8.541 Using the *length* Property of the *Window* Object

```
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to return the number of frames in the
// window passed.
function numFrames(win){
  return win.length;
}

// -->
</script>
```

window.location

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`window.location`

Description

The `location` property of the `Window` object returns the current URL of the document in the window.

Example

Listing 8.542 pops up an alert box that contains the URL of the current window.

Listing 8.542 Using the `location` Property of the `Window` Object

```
<script language="JavaScript" type="text/javascript">
<!--

// Display the current URL in an alert box.
alert(window.location);

// -->
</script>
```

window.locationbar

JavaScript1.2+
Nav4+

Syntax

`window.locationbar.property`

Description

The `locationbar` property of the `Window` object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the location bar is visible to the user.

NOTE

As of this writing, the `locationbar` property only has one subproperty: `visible`.

Example

See the example of `Window.locationbar.visible` for an example of using the `locationbar` property.

window.locationbar.visible

JavaScript1.2+
Nav4+

Syntax

window.locationbar.visible

Description

The `visible` subproperty of the `locationbar` property of the `Window` is used to determine whether the location bar is visible to the user. If it is visible, the property returns `true`. It returns `false` if the bar is not visible.

Example

Listing 8.543 determines whether several of the browser bars are displayed. In the example, you will see whether the location bar is visible by using the `visible` property.

Listing 8.543 Using the *visible* Property of `locationbar`

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.menubar

JavaScript1.2+
Nav4+

Syntax

window.menubar.property

Description

The `menubar` property of the `Window` object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the menu bar is visible to the user.

NOTE

As of this writing, the `menubar` property only has one subproperty: `visible`.

Example

See the example of `Window.menubar.visible` for an example of using the `menubar` property.

window.menubar.visible

JavaScript1.2+
Nav4+

Syntax

`window.menubar.visible`

Description

The `visible` subproperty of the `menubar` property of the `Window` is used to determine whether the menu bar is visible to the user. If it is visible, the property returns `true`. It returns `false` if the bar is not visible.

Example

Listing 8.544 determines whether several of the browser bars are displayed. In the example, you will see whether the menu bar is visible by using the `visible` property.

Listing 8.544 Using the *visible* Property of `menubar`

```
<script language="JavaScript" type="text/javascript">
<!--
```

```
// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');
```

```
// Close the stream to the document.  
document.close();  
  
// -->  
</script>
```

window.moveBy()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.moveBy(numHorz, numVert)

Description

The `moveBy()` method of the Window object moves the specified window by the number of pixels passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the number of horizontal pixels you want to move the window, whereas the second numeric value represents the vertical number of pixels.

If the numbers passed are positive, the window is moved to the right horizontally, and down vertically. Negative numbers move the window in the opposite direction.

Example

Listing 8.545 has four buttons: Up, Down, Right, and Left. If you click these buttons, the window the document is loaded in will move one pixel at a time in that direction.

Listing 8.545 *Using the `moveBy()` Method to Move the Location of a Window*

```
<html>  
<head>  
</head>  
<script language="JavaScript1.2" type="text/javascript">  
<!--
```

```
// Define a function to handle the window movement.  
function moveWin(dir, dist){
```

```
    // Define variables to hold the movement values.  
    var myVert;  
    var myHorz;
```

```
    // Determine the type of movement.  
    if(dir == "vert"){  
        myHorz = 0;
```

Listing 8.545 Continued

```
    myVert = dist;
  }else{
    myHorz = dist;
    myVert = 0;
  }

  // Move the window.
  window.moveBy(myHorz, myVert);
}
// -->
</script>
<body>
<form>
<table border=0>
  <tr>
    <td></td>
    <td><input type="button" value=" Up " onclick="moveWin('vert',-1)"></td>
    <td></td>
  </tr>
  <tr>
    <td><input type="button" value=" Left " onclick="moveWin('horz',-1)"></td>
    <td></td>
    <td><input type="button" value="Right" onclick="moveWin('horz',1)"></td>
  </tr>
  <tr>
    <td></td>
    <td><input type="button" value="Down" onclick="moveWin('vert',1)"></td>
    <td></td>
  </tr>
</table>
</form>
</body>
</html>
```

window.moveTo()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.moveTo(numX, numY)

Description

The `moveTo()` method of the `Window` object moves the specified window to the specified location passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the x coordinate to which you want to move the window, whereas the second numeric value represents the y coordinate.

Example

Listing 8.546 has two text fields and a button. If the user enters an integer value in each of the text fields and clicks the button, the window will move to that location.

Listing 8.546 *Using the `moveTo()` Method to Move the Location of a Window*

```
<html>
<head>
</head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window movement.
function moveWin(form){

    // Define variables to hold the movement values.
    var myX = form.X.value;
    var myY = form.Y.value;

    // Move the window.
    window.moveTo(myX, myY);
}
// -->
</script>
<body>
<form>
    <b>X-Coordinate:</b>
    <input type="text" name="X"><br>
    <b>Y-Coordinate:</b>
    <input type="text" name="Y"><br>
    <input type="button" value="Move Window" onclick="moveWin(this.form)"></td>
</form>
</body>
</html>
```

window.name

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`window.name`

Description

The `name` property of an instance of a `Window` object returns the name of the window. This property contains the name specified when new windows are created using the `Window.open()` method. In JavaScript 1.0, this property was read only, but this was

changed in JavaScript 1.1 so that you can assign a name to a window not created with the `Window.open()` method. This property was tainted in JavaScript 1.1 as well.

Example

Listing 8.547 has a button that launches a second window. The name of the window is written to it using the name property of the `Window` object.

Listing 8.547 Using the name Property to Retrieve the Name of a Window

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to open a small window.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizable=no';
    var myFeatures = myBars + ', ' + myOptions;

    // Open the window. Give the window instance the name newWin and
    // name the document in the window myDoc.
    var newWin = open('', 'myDoc', myFeatures);

    // Write the window's name to the new window.
    newWin.document.writeln('This window\'s name is: ' + newWin.name + '<br>');
    newWin.document.writeln('<form>');
    newWin.document.writeln('<input type="button" value="Close">');
    newWin.document.writeln(' onclick="window.close()">');
    newWin.document.writeln('</form>');

    // Close the stream to the document.
    newWin.document.close();

}
// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Open" onclick="openWin()">
</form>
</body>
</html>
```

window.offscreenBuffering

JavaScript1.2, JScript3.0+
Nav4+, IE4+

Syntax

`window.offscreenBuffering = boolean`

Description

The `offscreenBuffering` property of the Window object is used to explicitly instruct the browser whether to buffer data offscreen before displaying. Without doing this, a user's window might flicker as the page is being drawn. This property simply takes a *boolean* value of `true` or `false` to set it.

Example

Listing 8.548 shows how you can instruct the browser to not buffer data offscreen.

Listing 8.548 Using the `offscreenBuffering` Property

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Disallow off screen buffering
window.offscreenBuffering = false;

//-->
</script>
```

window.onBlur

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onblur="command"`

Description

The `onBlur` event handler is a property of a Window object and is fired when the focus is moved away from that particular window instance. Care should be taken when using this event handler because it is possible to get into an infinite loop when using the `onFocus` event handler or the `focus()` method. Note that when this event handler is called within the `<body>` tag, it is overridden if a `<frame>` tag that also uses this event handler loaded the document.

NOTE

Some Navigator 3 browsers do not fully support this event handler when called in a `<frameset>` tag.

Example

Listing 8.549 has a frame set with two frames. The first frame, `toc`, has the `onBlur` event handler specified in its tag. When focus leaves this frame, the event is fired and the `myBlurFunc()` function will be called.

Listing 8.549 Example of Using the `onBlur` Event

```
<frameset cols="150,*">
  <frame name="toc"
    src="/toc.htm"
    onBlur='myBlurFunc()'
    marginwidth="1" marginheight="1" scrolling="auto">
  <frame name="body"
    src="/body.htm"
    marginwidth="1" marginheight="5" scrolling="auto">
</frameset>
```

window.onDragDrop

JavaScript1.2+
Nav4+

Syntax

`ondragdrop="command"`

Description

The `onDragDrop` event handler of a property of a `Window` object is fired when the user drops an object, such as a file, on that particular window instance.

Example

In Listing 8.550, if you try to drop a new file on to the browser when this page is loaded, you will be asked to confirm this operation. If you accept, the page will load. If you cancel, the page will not load.

Listing 8.550 Example of Using the `onDragDrop` Event

```
<html>
<body ondragdrop='return(confirm("Are you sure you want to continue?"))'>
  Try to drop an element on this page.
</body>
</html>
```

window.onError

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

`onerror="command"`

Description

The `onError` event handler of the `Window` object is fired when an error occurs loading the page. You might find this useful to try and reload the page, using the `reload()` method of the `Location` object.

Example

Listing 8.551 is an example of placing the `onError` event handler in the `<body>` tag. If there is an error when loading this page, an alert box will be displayed to the user.

Listing 8.551 *Example of Using the `onError` Event Handler*

```
<body onerror='alert("Error: There has been an error loading this page.>")'>
```

window.onFocus

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

```
onfocus="command"
```

Description

The `onFocus` event handler of a property of a `Window` object is fired when the focus is placed on that particular window instance. Care should be taken when using this event handler because it is possible to get into an infinite loop when using `onBlur` event handler or `blur()` method. Note that when this event handler is called within the `<body>` tag, it is overridden if a `<frame>` tag that also uses this event handler loaded the document.

NOTE

Some Navigator 3 browsers do not fully support this event handler when called in a `<frameset>` tag.

Example

Listing 8.552 has a frame set with two frames. The first frame, `toc`, has the `onFocus` event handler specified in its tag. When the focus leaves this frame, the event is fired and the `myFocusFunc()` function (not shown) will be called.

Listing 8.552 *Example of Using the `onFocus` Event*

```
<frameset cols="150,*">
  <frame name="toc"
    src="/toc.htm"
    onfocus='myFocusFunc()'
    marginwidth="1" marginheight="1" scrolling="auto">
```

Listing 8.552 Continued

```
<frame name="body"
      src="/body.htm"
      marginwidth="1" marginheight="5" scrolling="auto">
</frameset>
```

window.onLoad

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`onload="command"`

Description

The `onLoad` event handler of a property of a `Window` object is fired when the page has finished loading in that particular window instance.

NOTE

The `onLoad` event handler in the `<body>` of a document that is loaded in a frame will fire before an event handler loaded in the `<frameset>` tag that loaded the document.

Example

Listing 8.553 pops up an alert box when the page has finished loading.

Listing 8.553 Example of Using the `onLoad` Event

```
<body onload='alert("The document has completely loaded.")'>
```

window.onMove

JavaScript1.2+
Nav4+

Syntax

`onmove="command"`

Description

The `onMove` event handler of a property of a `Window` object is fired when the window it is referenced in is moved. The user physically moving the window or a script moving it can fire this event.

Example

Listing 8.554 pops up an alert box if the user tries to move the window.

Listing 8.554 Using the onMove Event to Display an Alert Box

```
<body onmove='alert("Do NOT move this window!")'>
```

window.onResize

JavaScript1.2+
Nav4+

Syntax

onresize="command"

Description

The onResize event handler of a property of a Window object is fired when the window it is referenced in is resized. The user physically resizing the window or a script resizing it can fire this event.

Example

Listing 8.555 pops up an alert box if the user tries to resize the window.

Listing 8.555 Using the onResize Event to Display an Alert Box

```
<body onresize='alert("Do NOT resize this window!")'>
```

window.onUnload

JavaScript 1.0+, JScript 1.0+
Nav2+, IE3+, Opera3+

Syntax

onunload="command"

Description

The onUnload event handler of a property of a Window object is fired when the page is unloaded in that particular window instance. This occurs when the user leaves the page for another page.

NOTE

The onUnload event handler in the <body> of a document that is loaded in a frame will fire before an event handler loaded in the <frameset> tag that loaded the document.

Example

Listing 8.556 pops up an alert box when the user leaves the page.

Listing 8.556 Example of Using the onUnload Event

```
<body onunload='alert("Please do not leave!")'>
```

window.open()

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`window.open(pageURL, name, parameters)`

Description

The `open()` method of the `Window` object creates a new instance of a window. It loads the *pageURL* passed to the method in a window based on the *parameters* specified. The `action` attribute of the `<form>` tag and the `target` attribute of the `<a>` tag can reference the window by the *name* passed.

Most of the *parameters* passed, which are listed without spaces and commas, are toggled on and off by setting them to `yes` or `no`. It is also possible to use `1` or `0` to turn these features on or off. Either way, you should be consistent across each of the options. Table 8.42 has the different *parameters* that can be passed and how to turn them on and off.

NOTE

If you place spaces in the parameter string, the options will not work. Be sure to comma separate each of these options and do not insert any spaces.

Table 8.42 Parameters That Can Be Passed When Creating a New Instance of the Window Object Using the open() Method

Parameter	Initialize With	Description
<code>alwaysLowered</code>	<code>yes/no</code>	This parameter tells the window to stay behind all other windows. This must be done in signed scripts because it is a secure feature and was implemented in JavaScript 1.2.
<code>alwaysRaised</code>	<code>yes/no</code>	This parameter tells the window to stay on top of all other windows. This must be done in signed scripts because it is a secure feature and was implemented in JavaScript 1.2.
<code>dependent</code>	<code>yes/no</code>	This parameter opens the window as a true child window of the parent window. When the parent window is closed, so is the child window. This feature was implemented in JavaScript 1.2.
<code>directories</code>	<code>yes/no</code>	Specifies whether the directory bar on Navigator 2 and 3 is visible on the new window.

Parameter	Initialize With	Description
height	pixel value	Sets the height in pixels of the window. This feature, although still existent for backward compatibility, was removed in JavaScript 1.2 and replaced with <code>innerHeight</code> .
hotkeys	yes/no	Disables all but the Security and Quit hotkeys in a new window with no menu bar. This feature was implemented in JavaScript 1.2.
innerHeight	pixel value	Sets the height in pixels of the document in the window. This feature was implemented in JavaScript 1.2.
innerWidth	pixel value	Sets the width in pixels of the document in the window. This feature was implemented in JavaScript 1.2.
location	yes/no	Specifies whether the location bar is visible on the new window.
menubar	yes/no	Specifies whether the menu bar is visible on the new window.
outerHeight	pixel value	Sets the height in pixels of the window, including the chrome. This feature was implemented in JavaScript 1.2.
outerWidth	pixel value	Sets the width in pixels of the window, including the chrome. This feature was implemented in JavaScript 1.2.
resizable	yes/no	Specifies whether the window can be resized.
screenX	pixel value	Sets the distance in pixels of the window from the left side of the screen. This feature was implemented in JavaScript 1.2.
screenY	pixel value	Sets the distance in pixels of the window from the top of the screen. This feature was implemented in JavaScript 1.2.
scrollbars	yes/no	Specifies whether the scrollbars are visible on the new window.
titlebar	yes/no	Specifies whether the title bar is visible on the new window.
toolbar	yes/no	Specifies whether the toolbar is visible on the new window.
width	pixel value	Sets the width in pixels of the window. This feature, although still existent for backward compatibility, was removed in JavaScript 1.2 and replaced with <code>innerWidth</code> .


```

newWin.document.writeln('</td></tr>');
newWin.document.writeln('<tr valign="top"><td>');
newWin.document.writeln('<textarea cols="45" rows="7" wrap="soft">');
newWin.document.writeln(myReadme + '</textarea>');
newWin.document.writeln('</td></tr>');
newWin.document.writeln('<tr valign="bottom" align="right">');
newWin.document.writeln(' bgcolor="#c0c0c0"><td>');
newWin.document.writeln('<input type="button" value="Close">');
newWin.document.writeln(' onclick="window.close()">');
newWin.document.writeln('</td></tr>');
newWin.document.writeln('</table></form>');

// Close the stream to the document and bring the window to the front.
newWin.document.close();
newWin.focus();
}

// -->
</script>
<body>
<form>
  <b>Click the following button to open a new window: </b>
  <input type="button" value="Open" onclick='openWin()'>
</form>
</body>
</html>

```

window.opener

JavaScript1.1+, JScript1.0+
Nav3+, IE3+, Opera3+

Syntax

window.opener
window.opener.method
window.opener.property

Description

The `opener` property of the `Window` object corresponds to the window that opens the window from which the property was accessed. When accessed by a child window, it returns the parent window. With this property, you can then invoke methods and access properties of the `Window` object on the “opener” window. This property can also be set in scripts that allow the browser to clean up the reference to the parent window if it is closed before the child window. Most browsers have limits on the number of open windows they can have, and, by cleaning up these closed windows, you are able to regain the ability to open more windows if your limit has been reached. This is accomplished by setting the `opener` property to `null`.

Example

Listing 8.558 has a button that opens a second window when clicked. In the second window, there is a button that closes the parent window by referencing it via the opener property. After the `close()` method has been called on this window, the opener property is set to `null` to clean up the parent window.

Listing 8.558 Using the opener Property to Return the Parent Window

```
<html>
<head>

<script language="JavaScript" type="text/javascript">
<!--

// Define the openWin() function.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ',titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ', ' + myOptions;
    var myReadme = "Welcome to Pure JavaScript!\n" +
        "-----\n" +
        'You can enter some text here.'

    // Open the window. Give the window instance the name newWin and
    // name the document in the window myDoc.
    var newWin = open('', 'myDoc', myFeatures);

    newWin.document.writeln('<form>');
    newWin.document.writeln('<table cellpadding="0" cellspacing="0"
border="1">');
    newWin.document.writeln('<tr valign="top" bgcolor="#000099"><td>');
    newWin.document.writeln('<font size="-1" color="#ffffff"><b>');
    newWin.document.writeln('&nbsp;&nbsp;&nbsp;Readme</b></font>');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('<tr valign="top"><td>');
    newWin.document.writeln('<textarea cols="45" rows="7" wrap="soft">');
    newWin.document.writeln(myReadme + '</textarea>');
    newWin.document.writeln('</td></tr>');
    newWin.document.writeln('<tr valign="bottom" align="right" >');
    newWin.document.writeln('<bgcolor="#c0c0c0"><td>');

    // Close the opener window and clean it up
    newWin.document.writeln('<input type="button" value="Close">');
    var myJS = "window.opener.close();window.opener=null"
    newWin.document.writeln('onclick=" ' + myJS + ' ">');
}
```



```

newWin.document.writeln('</td></tr>');
newWin.document.writeln('</table></form>');

// Close the stream to the document and bring the window to the front.
newWin.document.close();
newWin.focus();
}

// -->
</script>
</head>
<body>
<form>
  <b>Click the following button to open a new window: </b>
  <input type="button" value="Open" onclick='openWin()'>
</form>
</body>
</html>

```

window.outerHeight

JavaScript1.2+
Nav4+

Syntax

window.outerHeight

Description

The `outerHeight` property of the `Window` object references the pixel height of the browser's frame. This includes any of the toolbars or other "chrome" that makes up the frame itself.

Example

Listing 8.559 has a button that, when clicked, opens up a second, smaller window. The `outerHeight` property is written to this new window.

Listing 8.559 Using the *outerHeight* Property

```

<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to open a small window.
function openWin(){

    // Create variables to hold the various options that can be set

```

Listing 8.559 Continued

```
// when a new Window instance is created.
var myBars = 'directories=no,location=no,menubar=no,status=no';
myBars += ',titlebar=no,toolbar=no';
var myOptions = 'scrollbars=no,width=400,height=200,resizable=no';
var myFeatures = myBars + ',' + myOptions;

// Open the window. Give the window instance the name newWin and
// name the document in the window myDoc.
var newWin = open('', 'myDoc', myFeatures);

// Write the window height and width properties to the new window.
newWin.document.writeln('<h4>Properties for this Window</h4>');
newWin.document.writeln('innerHeight: ' + newWin.innerHeight + '<br>');
newWin.document.writeln('innerWidth: ' + newWin.innerWidth + '<br>');
newWin.document.writeln('outerHeight: ' + newWin.outerHeight + '<br>');
newWin.document.writeln('outerWidth: ' + newWin.outerWidth + '<br>');
newWin.document.writeln('<form>');
newWin.document.writeln('<input type="button" value="Close">');
newWin.document.writeln(' onclick="window.close()">');
newWin.document.writeln('</form>');

// Close the stream to the document.
newWin.document.close();

}
// -->
</script>
</head>
<body>
<form>
  <input type="button" value="Open" onclick="openWin()">
</form>
</body>
</html>
```

window.outerWidth

JavaScript1.2+
Nav4+

Syntax

`window.outerWidth`

Description

The `outerWidth` property of the `Window` object references the pixel width of the browser's frame. This includes any of the toolbars or other "chrome" that make up the frame itself.

Example

Listing 8.560 has a button that, when clicked, opens up a second, smaller window. The `outerWidth` property is written to this new window.

Listing 8.560 *Using the `outerWidth` Property*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to open a small window.
function openWin(){

    // Create variables to hold the various options that can be set
    // when a new Window instance is created.
    var myBars = 'directories=no,location=no,menubar=no,status=no';
    myBars += ' ,titlebar=no,toolbar=no';
    var myOptions = 'scrollbars=no,width=400,height=200,resizeable=no';
    var myFeatures = myBars + ' , ' + myOptions;

    // Open the window. Give the window instance the name newWin and
    // name the document in the window myDoc.
    var newWin = open(' ', 'myDoc', myFeatures);

    // Write the window height and width properties to the new window.
    newWin.document.writeln('<h4>Properties for this Window</h4>');
    newWin.document.writeln('innerHeight: ' + newWin.innerHeight + '<br>');
    newWin.document.writeln('innerWidth: ' + newWin.innerWidth + '<br>');
    newWin.document.writeln('outerHeight: ' + newWin.outerHeight + '<br>');
    newWin.document.writeln('outerWidth: ' + newWin.outerWidth + '<br>');
    newWin.document.writeln('<form>');
    newWin.document.writeln('<input type="button" value="Close">');
    newWin.document.writeln(' onclick="window.close()">');
    newWin.document.writeln('</form>');

    // Close the stream to the document.
    newWin.document.close();

}
// -->
</script>
</head>
<body>
<form>
    <input type="button" value="Open" onclick="openWin()">
</form>
</body>
</html>
```

window.pageXOffset

JavaScript1.2+, JScript3+
Nav4+, IE4+

Syntax

`window.pageXOffset`

Description

The `pageXOffset` property of the `Window` object reflects the current horizontal pixel location of the top-left corner of the document in the window. In chromeless windows, this property can be referenced if you are moving a window with the `moveTo()` method before the actual move is made to see whether the window needs to be moved. It is also useful when using the `scrollTo()` method because it returns the current location of the viewable document in relation to the whole page.

Example

Listing 8.561 has a button that, when clicked, displays the current x and y coordinates of the window.

Listing 8.561 Using the `pageXOffset` Property to See the Current Location of the Window

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to display an alert box with the current
// window location.
function showLocation(){

    // Store the offset in variables.
    var x = self.pageXOffset;
    var y = self.pageYOffset

    // Build a string to display.
    var currX = "X-coordinate: " + x + "\n";
    var currY = "Y-coordinate: " + y;

    // Display the coordinates.
    window.alert(currX + currY);
}

-->
</script>
</head>
<body>
```

```
<form>
  <input type="button" value="Show Location" onclick="showLocation()">
</form>
</body>
</html>
```

window.pageYOffset

JavaScript1.2+, JScript3+
Nav4+, IE4+

Syntax

window.pageYOffset

Description

The `pageYOffset` property of the `Window` object reflects the current vertical pixel location of the top-left corner of the document in the window. In chromeless windows, this property can be referenced if you are moving a window with the `moveTo()` method before the actual move is made to see whether the window needs to be moved. It is also useful when using the `scrollTo()` method because it returns the current location of the viewable document in relation to the whole page.

Example

Listing 8.562 has a button that, when clicked, displays the current x and y coordinates of the window.

Listing 8.562 Using the `pageYOffset` Property to See the Current Location of the Window

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to display an alert box with the current
// window location.
function showLocation(){

    // Store the offset in variables.
    var x = self.pageXOffset;
    var y = self.pageYOffset

    // Build a string to display.
    var currX = "X-coordinate: " + x + "\n";
    var currY = "Y-coordinate: " + y;
```

Listing 8.562 Continued

```
// Display the coordinates.
window.alert(currX + currY);
}

// -->
</script>
</head>
<body>
<form>
  <input type="button" value="Show Location" onclick="showLocation()">
</form>
</body>
</html>
```

window.parent

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

```
window.parent.parent[num]
window.parent.frameName
```

Description

The parent property of the Window object contains a reference to the parent window of any frames that are loaded. In the instance where Frame A loads a page with a <frame-set> with Frame A.1 and A.2, the parent of the documents in A.1 and A.2 is Frame A. Frame A's parent is the top level window.

The referencing of these sibling frames can either be done using the frames array and passing an index number, or you can directly reference a frame using the name that is assigned by the name attribute of the <frame> tag.

Example

Listing 8.563 shows how to reference the parent of the third frame on a page.

Listing 8.563 Using the parent Property to Reference a Frame

```
var myFrameReference = myWin.parent.frames[2];
```

window.personalbar

JavaScript1.2+
Nav4+

Syntax

```
window.personalbar.property
```

Description

The `personalbar` property of the `Window` object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the personal bar is visible to the user.

NOTE

As of this writing, the `personalbar` property only has one subproperty: `visible`.

Example

Listing 8.564 determines whether several of the browser bars are displayed. In the example, you will see whether the personal bar is visible by using the `visible` property.

Listing 8.564 Using the `visible` Property of `personalbar`

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menuBar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationBar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalBar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollBars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusBar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.personalbar.visible

JavaScript1.2+
Nav4+

Syntax

`window.personalbar.visible`

Description

The `visible` subproperty of the `personalbar` property of the `Window` is used to determine whether the personal bar is visible to the user. If it is visible, the property returns `true`. It returns `false` if the bar is not visible.

Example

Listing 8.565 determines whether several of the browser bars are displayed. In the example, you will see whether the personal bar is visible by using the `visible` property.

Listing 8.565 *Using the `visible` Property of `personalbar`*

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.print()

JavaScript 1.2+, JScript 3.0+
Nav4+, IE4+

Syntax

`window.print()`

Description

The `print()` method of the `Window` object simulates the user clicking the Print button on the browser. It tells the browser to open the Print dialog box to print the current page.

Example

Listing 8.566 has a button. Clicking the button will tell the browser to open the Print dialog box to allow the user to print the current page.

Listing 8.566 *Using the `print()` Method to Print the Current Page*

```
<html>
<body>
<form>
```



```


</form>
</body>
</html>

```

window.prompt()

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

`window.prompt(string1, string2)`

Description

The `prompt()` method of the `Window` object displays a prompt dialog box when invoked. The value of *string1* passed to the method is displayed in the box, and the value of *string2* is contained in the text field of the prompt dialog box. The returned value of this method is the text in the text field.

Example

Listing 8.567 pops up a prompt box when the script is loaded, asking the user for a password. If the correct password is entered, the page finishes loading. The result of running this script can be seen in Figure 8.6.

Listing 8.567 A Prompt Box Displayed Using the `prompt()` Method

```

<script language="JavaScript" type="text/javascript">
<!--

// Keep asking the user for a password until they get it right.
while(prompt('Please enter your password', 'HERE') != 'admin'){
    alert('That was an incorrect response, please try again');
}

// This is only executed if 'admin' is entered.
document.write('You have entered the correct password!');

// -->
</script>

```

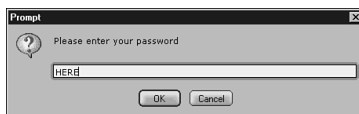


Figure 8.6

An alert box created with the `Window.prompt()` method.

window.releaseEvents()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
window.releaseEvents(event)  
window.releaseEvents(event1 | event2 | eventN)
```

Description

The `releaseEvents()` method of the `Window` object releases all previously captured events of the event type passed. These events can be captured with the `Window.captureEvents()` method. The events that can be released are as follows:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`
- `Event.MOUSEOVER`
- `Event.MOUSEUP`
- `Event.MOVE`
- `Event.RESET`
- `Event.RESIZE`
- `Event.SELECT`
- `Event.SUBMIT`
- `Event.UNLOAD`

After one of these events has been captured, you can define a function to replace the built-in method for handling the event. Use the `releaseEvents()` method to free the event after a capture.

Example

Listing 8.568 has a single text box and a button. The script in the `<head>` of the document specifies a function to handle all `onClick` events in the window. To be able to do this, the `captureEvents()` method has to be used to capture all events of type

Event.CLICK. When the page itself is clicked, a counter, which is displayed in the text box, is incremented.

When the button is pressed down, the onMouseDown event handler is fired and the Event.CLICK is released and no longer increments the page when the page is clicked.

Listing 8.568 Capturing Events with the `Window.releaseEvents()` **Method**

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK);
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text box.
    window.document.myForm.myText.handleEvent(e);
}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}

// Releases the click event capturing.
function releaseClick(){
    window.releaseEvents(Event.CLICK);
}

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText" onclick='changeText()'>
    <input type="button" value="Release Event" onmousedown='releaseClick()'>
</form>
</body>
</html>
```

window.resizeBy()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
window.resizeBy(numHort, numVert)
```

Description

The `resizeBy()` method of the `Window` object resizes the specified window by the number of pixels passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the number of vertical pixels you want to size the window by, whereas the second numeric value represents the horizontal number of pixels.

If the numbers passed are positive, the window size is increased. Negative numbers reduce the size of the window.

Example

Listing 8.569 has four buttons. Two buttons are for increasing height, and the other two are for increasing width. If you click these buttons, the window will resize 10 pixels at a time.

Listing 8.569 *Using the `resizeBy()` Method to Resize a Window*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window resizing.
function resizeWin(dir, dist){

    // Define variables to hold the sizing values.
    var myVert;
    var myHorz;

    // Determine the type of movement.
    if(dir == "vert"){
        myHorz = 0;
        myVert = dist;
    }else{
        myHorz = dist;
        myVert = 0;
    }

    // Resize the window.
    window.resizeBy(myHorz, myVert);
}
```

```
// -->
</script>
</head>
<body>
<form>
<table border=0>
  <tr>
    <td colspan="2">
      <input type="button" value="Expand Down"
        onclick="resizeWin('vert',10)">
    </td>
  </tr>
  <tr>
    <td>
      <input type="button" value="Retract From Right"
        onclick="resizeWin('horz',-10)">
    </td>
    <td>
      <input type="button" value="Grow Right"
        onclick="resizeWin('horz',10)">
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="button" value="Retrack Up"
        onclick="resizeWin('vert',-10)">
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

window.resizeTo()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.resizeTo(numWidth, numHeight)

Description

The `resizeTo()` method of the `Window` object resizes the specified window to the specified size passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the width you want to size the window to, whereas the second numeric value represents the height.

Example

Listing 8.570 has two text fields and a button. If the user enters an integer value in each of the text fields and clicks the button, the window will resize to those settings.

Listing 8.570 *Using the `resizeTo()` Method to Resize the Window*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window resizing.
function resizeWin(form){

    // Define variables to hold the resize values.
    var myWidth = form.width.value;
    var myHeight = form.height.value;

    // Resize the window.
    window.resizeTo(myWidth, myHeight);
}
// -->
</script>
</head>
<body>
<form>
    <b>New Width:</b>
    <input type="text" name="width"><br>
    <b>New Height:</b>
    <input type="text" name="height"><br>
    <input type="button" value="Resize Window"
        onclick="resizeWin(this.form)"></td>
</form>
</body>
</html>
```

window.routeEvent()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

`window.routeEvent(event)`

Description

The `routeEvent()` method of the Window object passes all previously captured events of the event type passed through their normal event process. The events that can be passed are as follows:

- `Event.ABORT`
- `Event.BLUR`
- `Event.CHANGE`
- `Event.CLICK`
- `Event.DBLCLICK`
- `Event.DRAGDROP`
- `Event.ERROR`
- `Event.FOCUS`
- `Event.KEYDOWN`
- `Event.KEYPRESS`
- `Event.KEYUP`
- `Event.LOAD`
- `Event.MOUSEDOWN`
- `Event.MOUSEMOVE`
- `Event.MOUSEOUT`
- `Event.MOUSEOVER`
- `Event.MOUSEUP`
- `Event.MOVE`
- `Event.RESET`
- `Event.RESIZE`
- `Event.SELECT`
- `Event.SUBMIT`
- `Event.UNLOAD`

After one of these events has been captured using the `Window.captureEvents()` method, you can define a function to replace the built-in method for handling the event. Use the `releaseEvents()` method to free the event after a capture, and use `routeEvent()` to allow the normal processing to take place.

Example

Listing 8.571 has a single text box and a link. The script in the `<head>` of the document specifies a function to handle all `onClick` events in the window. To be able to do this, the `captureEvents()` method has to be used to capture all events of type `Event.CLICK`. When the page itself is clicked, a counter, which is displayed in the text box, is incremented.

When the link is clicked, the `onMouseDown` event handler is fired and the `Event.CLICK` is routed through its normal means and no longer increments the page when the page is clicked.

Listing 8.571 *Capturing Events with the `Window.routeEvent()` Method*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--
```

Listing 8.571 Continued

```
// Define a click counter variable.
var counter = 0;

// Tell the browser you want to intercept ALL click events
// on the page. Then define a function to handle them.
window.captureEvents(Event.CLICK);
window.onClick = myClickHandler;

// Define the myClickHandler function to handle click events.
function myClickHandler(e){

    // Pass all click events to the onClick event of the text box.
    window.document.myForm.myText.handleEvent(e);
}

// Function is called by onClick of text box. Displays the number
// of clicks that have occurred.
function changeText(){
    document.myForm.myText.value = counter++;
}

// Releases the click event capturing.
function releaseClick(){
    window.routeEvent(Event.CLICK);
}

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText" onclick='changeText()'>
    <a href="http://www.purejavascript.com"
        onmousedown='window.routeEvent(Event.CLICK) '>Click Here!</a>
</form>
</body>
</html>
```

window.screenX

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.screenX

Description

The `screenX` property of the `Window` object is used to set the x coordinate of the left edge of the window. Within Netscape browsers, the property requires the `UniversalBrowserWrite` privilege.

Example

Listing 8.572 simply pops up an alert box that contains the `screenX` property value.

Listing 8.572 *Checking the `screenX` Property Value*

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
    alert(window.screenX)
    //-->
  </script>
</body>
</html>
```

window.screenY

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

`window.screenY`

Description

The `screenY` property of the `Window` object is used to set the y coordinate of the left edge of the window. Within Netscape browsers, the property requires the `UniversalBrowserWrite` privilege.

Example

Listing 8.573 simply pops up an alert box that contains the `screenY` property value.

Listing 8.573 *Checking the `screenY` Property Value*

```
<html>
<body>
  <script type="text/javascript" language="JavaScript1.2">
    <!--
    alert(window.screenY)
    //-->
  </script>
</body>
</html>
```

window.scroll()

JavaScript1.1, JScript3.0

Nav3, IE3, Opera3

Syntax

```
window.scroll(numX, numY)
```

Description

The `scroll()` method of the `Window` object scrolls the specified window to the specified location passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the x coordinate to which you want to scroll the window, whereas the second numeric value represents the y coordinate. Note that this method has been deprecated in JavaScript 1.2 and replaced with the `scrollBy()` and `scrollTo()` methods.

Example

Listing 8.574 has two text fields and a button. If the user enters an integer value in each of the text fields and clicks the button, the window will be scrolled to those settings.

Listing 8.574 Using the `scroll()` Method to Scroll the Window

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window scrolling.
function scrollWin(dir, dist){

    // Define variables to hold the scrolling values.
    var myVert;
    var myHorz;

    // Determine the type of scrolling.
    if(dir == "vert"){
        myHorz = 0;
        myVert = dist;
    }else{
        myHorz = dist;
        myVert = 0;
    }

    // Scroll the window.
    window.scroll(myHorz, myVert);
}
// -->
</script>
```

```
</head>
<body>
<form>
<table border=0>
  <tr>
    <td colspan="2">
      <input type="button" value="Down" onclick="scrollWin('vert',10)">
    </td>
  </tr>
  <tr>
    <td>
      <input type="button" value=" Left " onclick="scrollWin('horz',-10)">
    </td>
    <td>
      <input type="button" value="Right" onclick="scrollWin('horz',10)">
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="button" value=" Up " onclick="scrollWin('vert',-10)">
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

window.scrollbars

JavaScript1.2+
Nav4+

Syntax

window.scrollbars.property

Description

The `scrollbars` property of the `Window` object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the scrollbars are visible to the user.

NOTE

As of this writing, the `scrollbars` property only has one subproperty: `visible`.

Example

See the example of `Window.scrollbars.visible` for an example of using the `scrollbars` property.

window.scrollbars.visible

JavaScript1.2+
Nav4+

Syntax

`window.scrollbars.visible`

Description

The `visible` subproperty of the `scrollbars` property of the `Window` is used to determine whether the scrollbars are visible to the user. If they are visible, the property returns `true`. It returns `false` if the bars are not visible.

Example

Listing 8.575 determines whether several of the browser scrollbars are displayed. In the example, you will see whether the scrollbars are visible by using the `visible` property.

Listing 8.575 Using the *visible* Property of `scrollbars`

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's toolbar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.scrollBy()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

`window.scrollBy(numHorz, numVert)`

Description

The `scrollBy()` method of the `Window` object scrolls the specified window by the number of pixels passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the number of horizontal pixels by which you want to scroll the window, whereas the second numeric value represents the vertical number of pixels.

If the numbers passed are positive, the window is scrolled up. Negative numbers are scrolled down.

Example

Listing 8.576 has four buttons. Each of these buttons scroll the windows contents in different directions when clicked.

Listing 8.576 *Using the `scrollBy()` Method to Resize a Window*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window scrolling.
function scrollWin(dir, dist){

    // Define variables to hold the scrolling values.
    var myVert;
    var myHorz;

    // Determine the type of scrolling.
    if(dir == "vert"){
        myHorz = 0;
        myVert = dist;
    }else{
        myHorz = dist;
        myVert = 0;
    }

    // Scroll the window.
    window.scrollBy(myHorz, myVert);
}
// -->
</script>
</head>
<body>
<form>
<table border=0>
  <tr>
    <td colspan="2">
```

Listing 8.576 Continued

```
<input type="button" value="Down"
      onclick="scrollWin('vert',10)">
</td>
</tr>
<tr>
  <td>
    <input type="button" value=" Left "
          onclick="scrollWin('horz',-10)">
  </td>
  <td>
    <input type="button" value="Right"
          onclick="scrollWin('horz',10)">
  </td>
</tr>
<tr>
  <td colspan="2">
    <input type="button" value=" Up "
          onclick="scrollWin('vert',-10)">
  </td>
</tr>
</table>
</form>
</body>
</html>
```

window.scrollTo()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

window.scrollTo(numX, numY)

Description

The `scrollTo()` method of the `Window` object scrolls the specified window to the specified location passed to the method. As shown in the syntax definition, the first numeric value passed to the method represents the x coordinate to which you want to scroll the window, whereas the second numeric value represents the y coordinate.

Example

Listing 8.577 has two text fields and a button. If the user enters an integer value in each of the text fields and clicks the button, the window will be scrolled to those settings.

Listing 8.577 Using the `scrollTo()` Method to Scroll the Window

```

<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Define a function to handle the window scrolling.
function scrollWin(form){

    // Define variables to hold the scroll values.
    var myX = form.X.value;
    var myY = form.Y.value;

    // Scroll the window.
    window.scrollTo(myX, myY);
}
// -->
</script>
</head>
<body>
<form>
    <b>X-Coordinate:</b>
    <input type="text" name="X"><br>
    <b>Y-Coordinate:</b>
    <input type="text" name="Y"><br>
    <input type="button" value="Scroll Window"
        onclick="scrollWin(this.form)"></td>
</form>
</body>
</html>

```

window.self

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

window.self.method
window.self.property

Description

The `self` property of the `Window` object contains a reference to the current window. This allows you to invoke functions or call properties on the current window without any confusion when multiple windows are displayed.

Example

Listing 8.578 shows how to close the current window through the `self` reference.

Listing 8.578 Using the `self` Property to Reference the Current Window

```
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to close the current window
function closeWin(){

    self.close()

// -->
</script>
```

window.setHotKeys()

JavaScript1.2
Nav4

Syntax

window.setHotKeys(boolean)

Description

The `setHotKeys()` method of the `Window` object enables or disables all hot keys within a window that do not have menus. It simply takes a *boolean* value to enable or disable this option. Passing `true` will enable the hot keys, whereas `false` will disable them.

Example

Listing 8.579 shows how you can disable hot keys for a given window.

Listing 8.579 Using the `setHotKeys()` Method

```
<script type="text/javascript" language="JavaScript1.2">
<!--
setHotKeys(false)
//-->
</script>
```

window.setInterval()

JavaScript1.2+, JScript3.0+
Nav4+, IE4+

Syntax

```
window.setInterval(expression, milliseconds)
window.setInterval(function, milliseconds)
window.setInterval(function, milliseconds, arg1, ..., argN)
```


Description

The `setInterval()` method of the `Window` object sets an interval to invoke the expression or function that is passed to the method. The expression or function is invoked after every elapse of the *milliseconds*. As shown in the syntax definition, it is possible to pass arguments to the function you want to invoke. This interval can be cleared by using the `clearInterval()` method.

Example

Listing 8.580 sets an interval in the `<head>` of the document that displays the current time in a text box on the page. The interval is set so that it only updates the time in the text box every five seconds. There is also a button on this page that allows you to clear the interval if you click it.

Listing 8.580 *Clearing an Interval with the `setInterval()` Method*

```
<html>
<head>
<script language="JavaScript1.2" type="text/javascript">
<!--

// Create a variable to hold a counter.
var counter = 1;

// Define a function to display the counter.
function startCounter(){
    document.myForm.myText.value = counter++;
}

// Define a function to stop the counter.
function stopCounter(){
    window.clearInterval(myInterval);
}

// Set the interval to call the function every 5 seconds.
var myInterval = window.setInterval("startCounter()", 5000)

// -->
</script>
</head>
<body onload="startCounter()">
<form name="myForm">
    <input type="text" size="2" value="" name="myText">
    <input type="button" value="Clear Interval" onclick="stopCounter()">
</form>
</body>
</html>
```

window.setResizable()

JavaScript1.2

Nav4

Syntax

`window.setResizable(boolean)`

Description

The `setResizable()` method of the `Window` object enables or disables a user's ability to resize a window. It simply takes a *boolean* value to perform this option. Passing `true` will enable the user to resize the window, whereas `false` will disable this feature.

Example

Listing 8.581 shows how you can disable the resizing of a window.

Listing 8.581 Using the `setResizable()` Method

```
<script type="text/javascript" language="JavaScript1.2">
<!--
setResizable(false)
//-->
</script>
```

window.setTimeout()

JavaScript1.0+, JScript1.0+

Nav2+, IE3+, Opera3+

Syntax

```
window.setTimeout(expression, milliseconds)
window.setTimeout(function, milliseconds)
window.setTimeout(function, milliseconds, arg1, ..., argN)
```

Description

The `setTimeout()` method of the `Window` object sets a timeout to invoke the expression or function that is passed to the method. The expression or function is invoked after the elapse of the *milliseconds*. As shown in the syntax definition, it is possible to pass arguments to the function you want to invoke. This timeout can be cleared by using the `clearTimeout()` method.

Example

Listing 8.582 has a button and text box. By default, the time will be displayed in the text box after five seconds. This is done using the `setTimeout()` method. If the button is clicked, a function is called that invokes the `clearTimeout()` method, preventing the time from being displayed in the text box.

Listing 8.582 Using the `setTimeout()` Method

```

<html>
<head>
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to show the time.
function showTime(){
    myTime = new Date();
    myTime = myTime.getHours() + ":" + myTime.getMinutes() + ":";
    myTime += myTime.getSeconds();
    document.myForm.myText.value = myTime;
}

// Define a function to stop the display of the time.
function stopTime(){
    window.clearTimeout(myTimeout);
}

// Set the interval to call the function after 5 seconds.
var myTimeout = window.setTimeout("showTime()", 5000)

// -->
</script>
</head>
<body>
<form name="myForm">
    <input type="text" size="2" value="" name="myText">
    <input type="button" value="Clear Timeout" onclick="stopTime()">
</form>
</body>
</html>

```

window.setZOptions()

JavaScript1.2

Nav4

Syntax`window.setZOptions(type)`**Description**

The `setZOptions()` method of the `Window` object specifies the z-order stacking behavior of a window. It takes a *type* to indicate how this stacking can occur. The following are the possible values of *type*. To set this property in Navigator, you need the `UniversalBrowserWrite` privilege.

- `alwaysLowered`—Creates new windows below other windows, whether it is active or not.
- `alwaysRaised`—Creates new windows on top of other windows, whether it is active or not.
- `z-lock`—Creates new windows that do not rise above other windows when activated.

Example

Listing 8.583 shows how you can set the z-order stacking of a window.

Listing 8.583 Using the `setZOptions()` Method

```
<script type="text/javascript" language="JavaScript1.2">
<!--

// Set to lower ordering
setZOptions(alwaysLowered)

//-->
</script>
```

window.status

JavaScript1.0+, JScript1.0+
Nav2+, IE3+, Opera3+

Syntax

`window.status = string`

Description

The `status` property of the `Window` object allows you to specify the message that is displayed in the status bar of the browser. Note that in JavaScript 1.1, this property was tainted. See Chapter 1 for more information on JavaScript security and data tainting.

NOTE

When setting the `Window.status` property, be sure that your function returns `true`. This also applies when setting the property within the body of an HTML tag.

Example

Listing 8.584 shows how you can set the status in the Status Bar by rolling over a link.

Listing 8.584 Setting the Status of a Page

```
<a href="http://www.purejavascript.com/book"
  onMouseOver="window.status='Please Visit Our Online Book!'; return true"
  onMouseOut="window.status='Document: Done'">
Click Here!</a>
```

window.statusbar

JavaScript1.2+
Nav4+

Syntax

window.statusbar.property

Description

The `statusbar` property of the `Window` object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the status bar is visible to the user.

 NOTE

As of this writing, the `statusbar` property only has one subproperty: `visible`.

Example

See the example of `Window.statusbar.visible` for an example of using the `statusbar` property.

window.statusbar.visible

JavaScript1.2+
Nav4+

Syntax

window.statusbar.visible

Description

The `visible` subproperty of the `statusbar` property of the `Window` is used to determine whether the status bar is visible to the user. If it is visible, the property returns `true`. It returns `false` if the bar is not visible.

Example

Listing 8.585 determines whether several of the browser bars are displayed. In the example, you will see whether the status bar is visible by using the `visible` property.

Listing 8.585 Using the visible Property of statusbar

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.stop()

JavaScript1.2+, Jscript3.0
Nav4+, IE4+

Syntax

window.stop()

Description

The `stop()` method of the Window object simulates the user clicking the Stop button on the browser. It stops the browser from downloading and rendering the current page.

Example

Listing 8.586 has a button and an image reference to a nonexistent image. The browser will continue to try and download the image until it times out or the download is stopped. Clicking the button will stop the download.

Listing 8.586 Using the stop() Method to Stop a Page from Loading

```
<html>
<body>
<form>
  <input type="button" value="Stop" onclick="window.stop()">
</form>
<p>
  <table border="1" color= bgcolor="#FF0000">
    <tr>
      <td>
```

```

</td>
</tr>
</table>
</p>
</body>
</html>
```

window.toolbar

JavaScript1.2+
Nav4+

Syntax

window.toolbar.property

Description

The toolbar property of the Window object is, to some degree, an object itself. The real use of this property is to access its `visible` property to determine whether the toolbar is visible to the user.

NOTE

As of this writing, the `toolbar` property only has one subproperty: `visible`.

Example

See the example of `Window.toolbar.visible` for an example of using the toolbar property.

window.toolbar.visible

JavaScript1.2+
Nav4+

Syntax

window.toolbar.visible

Description

The `visible` subproperty of the `toolbar` property of the Window object is used to determine whether the toolbar is visible to the user. If it is visible, the property returns `true`. It returns `false` if the bar is not visible.

Example

Listing 8.587 determines whether several of the browser bars are displayed. In the example, you will see whether the toolbar is visible by using the `visible` property.

Listing 8.587 Using the visible Property of toolbar

```
<script language="JavaScript" type="text/javascript">
<!--

// Write the browser's bar status to the page. If the value
// is true, then it is displayed.
document.writeln('<h3>Browser Chrome Status</h3>')
document.writeln('Menu Bar: ' + window.menubar.visible + '<br>');
document.writeln('Tool Bar: ' + window.toolbar.visible + '<br>');
document.writeln('Location Bar: ' + window.locationbar.visible + '<br>');
document.writeln('Personal Bar: ' + window.personalbar.visible + '<br>');
document.writeln('Scroll Bars: ' + window.scrollbars.visible + '<br>');
document.writeln('Status Bar: ' + window.statusbar.visible + '<br>');

// Close the stream to the document.
document.close();

// -->
</script>
```

window.top

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

```
window.top.frames[num]
window.top.frameName
window.top.method
window.top.property
```

Description

The top property of the Window object contains a reference to the topmost browser window of any frames or pages that are loaded. In the instance where a Frame A loads a page with a <frameset> with Frame A.1 and A.2, the top of the documents in A.1 and A.2 is the window that actually has Frame A loaded. Frame A's top is also this window.

As shown in the syntax definition, the referencing of sibling frames can either be done using the frames array and passing an index number, or you can directly reference a frame using the name that is assigned by the name attribute of the <frame> tag. From within the current page or any of the frames, you can reference the top window and execute any methods or reference any properties that might reside there.

Example

Assuming that the page with this script lies within a <frameset>, Listing 8.588 shows how you can call a function that is defined in the topmost page.

Listing 8.588 Using the `top` Property to Call a Function in the Top Frame

```
<script language="JavaScript" type="text/javascript">
<!--

// Call a function in the top.
top.myFunc(myVar1, myVar2);

// -->
</script>
```

window.unwatch()

JavaScript 1.2+
Nav4+

Syntax

```
window.unwatch(property)
```

Description

The `unwatch()` method of the Window object is used to turn off the watch for a particular *property* specified by *property*.

Example

Listing 8.589 shows how the `unwatch()` method is used to stop watching the `outerHeight` property of the Window object after its value has changed.

Listing 8.589 Example of the `unwatch()` method of the Window object

```
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id, oldValue, newValue){
    document.write("ID (" + id + ") changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

//Start watch.
window.watch("outerHeight", alertme);

// Change value.
window.outerHeight = 100;

// End watch.
window.unwatch("outerHeight");
```

Listing 8.589 Continued

```
// -->
</script>
```

window.watch()

JavaScript 1.2+
Nav4+

Syntax

window.watch(property)

Description

The `watch()` method of the Window object is used to turn on the watch for a particular *property* specified by *property*.

Example

Listing 8.590 shows how the `watch()` method is used to start watching the `outerHeight` property of the Window object after its value has changed.

Listing 8.590 Example of the `watch()` method of the Window object

```
<script type="text/javascript" language="JavaScript">
<!--

function alertme(id, oldValue, newValue){
    document.write("ID (" + id + ") changed from " + oldValue + " to ");
    document.write(newValue + "<br>");
    return newValue;
}

//Start watch.
window.watch("outerHeight", alertme);

// Change value.
window.outerHeight = 100;

// -->
</script>
```

window.window

JavaScript1.0+, JScript1.0
Nav2+, IE3+, Opera3+

Syntax

`window.method`
`window.property`

Description

The window property of the Window object contains a reference to the current window. This allows you to invoke functions or call properties on the current window without any confusion when multiple windows are displayed.

Example

Listing 8.591 shows how to close the current window through the window reference.

Listing 8.591 Using the window Property to Reference the Current Window

```
<script language="JavaScript" type="text/javascript">
<!--

// Define a function to close the current window.
function closeWin(){

    window.close()

// -->
</script>
```




CHAPTER 9

Server-Side

This chapter is a detailed reference of all the items and elements making up the server-side JavaScript language. This refers to the implementation available in the Netscape and iPlanet Web Server Enterprise Edition, which we will simply refer to as Enterprise Server (ES) from this point on, and within Microsoft's Active Server Pages (ASP) environment, which is available for their Internet Information Server (IIS) or Personal Web Server (PWS).

Because both Microsoft and Netscape have taken different approaches for their server-side implementation, be sure to check the supported environments for these entries. Additionally, be sure to read Chapter 5, "JavaScript on the Server-Side," for more information on how these implementations are used.

As in other chapters of Part III, the details of the language are covered in this chapter. Each entry includes the language version, syntax, description, and an example of each server-side-specific language element.

The chapter is in alphabetical order, by JavaScript objects, to provide you with quick, easy access to the methods, properties, functions, and event handlers of every server-side object. These appear alphabetically after the respective parent object using simple dot notation.

addClient()

ES2+

Syntax

`addClient(URL)`

Description

The `addClient()` function is a top-level function that is not associated with any core object. This function is used to preserve the property values of a `client` object when you generate dynamic links or use the `redirect()` function. The `addClient()` function takes a *URL* as its only parameter.

Example

This example demonstrates how you can use the `addClient()` function when dynamically building links. In Listing 9.1, a link is built by using a property of the `project` object.

Listing 9.1 Using the `addClient()` Function to Dynamically Build a Link

```
<a href='<server>addClient("/myApp/page" + project.pagenum +  
".html")</server>'>  
Please proceed to the next page</a>
```

In Listing 9.2, the `addClient()` function is used in conjunction with the `redirect()` function. This will send the user to the URL specified in the `addClient()` function.

Listing 9.2 Using the `addClient()` Function with the `redirect()` Function

```
<server>  
  
// Check to see if the browser is Internet Explorer.  
if(request.agent.indexOf('MSIE') != -1){  
    redirect(addClient("/iepages/index.html"));  
  
// Redirect to another page if it is not IE.  
}else{  
    redirect(addClient("/defaultpages/index.html"));  
}  
</server>
```

addResponseHeader()

ES3+

Syntax

`addResponseHeader(key, value)`

Description

The `addResponseHeader()` function is a top-level function that is not associated with any core object. This function is used to add fields and values to the HTTP header sent back to the client. Because of when the actual header is sent in relation to the body of the data, you should be sure to set these fields and values before you call the `flush()` or `redirect()` functions.

NOTE

Because the JavaScript runtime engine flushes the output buffer after 64KB of content has been generated, you should be sure to call the `addResponseHeader()` function before this time.

Example

Listing 9.3 shows how you can send back a dynamically built external JavaScript source file to a browser with the proper content-type header field and value.

Listing 9.3 *Using the `addResponseHeader()` Function to Set the content-type of a File Being Sent to a Browser*

```
<server>

// Add a field to the header
addResponseHeader("content-type", "application/x-javascript");

</server>
```

Application

ASP1+

Syntax

Core ASP environment object

Description

The Application object is a core ASP environment object. It is used to share information across all users of a given application, which includes all .asp files as well as any virtual directories and their subdirectories. Table 9.1 contains the methods, collections, and events of this object.

Table 9.1 *Methods, Collections, and Events of the Application Object*

Type	Item	Description
Collection	Contents	Contains all items added to the application through script commands.
	StaticObjects	Contains all objects added to session with the <code><object></code> tag.

Table 9.1 *Continued*

Type	Item	Description
Event	Application_OnEnd	Occurs when the application quits.
	Application_OnStart	Occurs before the first new session is created.
Method	Contents.Remove()	Deletes an item from the Application object's Contents collection.
	Contents.RemoveAll()	Deletes all items from the Application object's Contents collection.
	Lock()	Prevents other clients from modifying Application object properties.
	Unlock()	Allows other clients to modify Application object properties.

Example

Listing 9.4 shows how you could specify the version of your application that users are implementing, and then write it out.

Listing 9.4 *Using the Application Object*

```
<script runat="server" type="text/jscript" language="JScript">
Application("version") = "1.0"
var appVer = Application("version");
Response.Write(appVer);
</script>
```

Application.Application_OnEnd

ASP1+

Syntax

```
function Application_OnEnd(){
    code
}
```

Description

The Application_OnEnd event function of the Application object is called when the Application_OnEnd event is fired. This occurs when the application quits, which is after the Session_OnEnd event is fired.

Example

Listing 9.5 shows how you can append a string to the end of the log file of the last request of an application.

Listing 9.5 Using the Application_OnEnd Event

```
<script runat="server" type="text/jscript" language="JScript">
function Application_OnEnd(){
    Response.AppendToLog("Your application has ended");
}
</script>
```

Application.Application_OnStart

ASP1+

Syntax

```
function Application_OnStart(){
    code
}
```

Description

The Application_OnStart event function of the Application object is called when the Application_OnStart event is fired. This occurs when the application starts, which is before the Session_OnStart event is fired.

Example

In Listing 9.6 shows how you can append a string to the end of the log file on the first request of an application.

Listing 9.6 Using the Application_OnStart Event

```
<script runat="server" type="text/jscript" language="JScript">
function Application_OnStart(){
    Response.AppendToLog("Your application has started");
}
</script>
```

Application.Contents

ASP1+

Syntax

```
Application.Contents(name)
```

Description

The Contents collection of the Application object contains all the items that have been added to the application through script commands. You are able to access a specific item by specifying its *name*.

Example

Listing 9.7 shows how you can use the `Remove()` method of the `Contents` collection to remove *myItem*, which was added to the application.

Listing 9.7 *Using the Contents Collection*

```
<script runat="server" type="text/jscript" language="JScript">

// Add the item to the collection.
Application("myItem") = "My name is Allen";

// Remove the item.
Application.Contents.Remove("myItem");
</script>
```

Application.Contents.Remove()

ASP1+

Syntax

```
Application.Contents.Remove(name)
Application.Contents.Remove(num)
```

Description

The `Remove()` method of the `Contents` collection of the `Application` object is used to remove an item from the application's collection. This item can either be referenced by its *name* or its *num* index position.

Example

Listing 9.8 shows how you can use the `Remove()` method of the `Contents` collection to remove *myItem*, which was added to the application.

Listing 9.8 *Using the Remove() Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Add the item to the collection.
Application("myItem") = "My name is Allen";

// Remove the item.
Application.Contents.Remove("myItem");
</script>
```

Application.Contents.RemoveAll()

ASP1+

Syntax

```
Application.Contents.RemoveAll()
```

Description

The `RemoveAll()` method of the `Contents` collection of the `Application` object is used to remove all items from the application's collection.

Example

Listing 9.9 shows how you can use the `RemoveAll()` method of the `Contents` collection to remove all items that have been added to the application.

Listing 9.9 *Using the `RemoveAll()` Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Add two items to the collection.
Application("myItem") = "My name is Allen";
Application("yourItem") = "Your name is Reader";

// Remove all items.
Application.Contents.RemoveAll();
</script>
```

Application.Lock()

ASP1+

Syntax

`Application.Lock()`

Description

The `Lock()` method of the `Application` object prevents other clients from modifying any items stored in the `Application` object. To unlock the object, you must call the `Unlock()` method, or the server will unlock it either when the page has finished processing or times out.

Example

In Listing 9.10 we lock a counter item before updating it. This item could be used to count the number of users a given application has serviced.

Listing 9.10 *Using the `Lock()` Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Initialize the counter, which should ONLY be done on the first page of
// the application.
Application("counter") = 0;

// Lock the application before incrementing the counter.
Application.Lock()
```

Listing 9.10 Continued

```
// Increment the counter.
Application("counter") += 1;

// Unlock the counter for others to modify.
Application.Unlock();
</script>
```

Application.StaticObjects

ASP1+

Syntax

`Application.StaticObjects(name)`

Description

The `StaticObjects` collection of the `Application` object stores all the objects created with `<object>` elements on a given ASP page. These are referenced by the *name* given to each instance of the elements.

Example

Listing 9.11 shows how you can grab the instance *myControl1* from your current ASP page.

Listing 9.11 Using the StaticObjects Collection

```
<script runat="server" type="text/jscript" language="JScript">
Application.StaticObjects("myControl1");
</script>
```

Application.Unlock()

ASP1+

Syntax

`Application.Unlock()`

Description

The `Unlock()` method of the `Application` object unlocks the previously locked `Application` object. To first lock the object, you must call the `Lock()` method. If you fail to call the `Unlock()` method, the server will unlock it when the page has finished processing or times out.

Example

In Listing 9.12 we lock a counter item before updating it. This item could be used to count the number of users a given application has serviced. After updating, we unlock it for others to access and modify.

Listing 9.12 Using the Unlock() Method

```

<script runat="server" type="text/jscript" language="JScript">

// Initialize the counter, which should ONLY be done on the first page of
// the application.
Application("counter") = 0;

// Lock the application before incrementing the counter.
Application.Lock()

// Increment the counter.
Application("counter") += 1;

// Unlock the counter for others to modify.
Application.Unlock();
</script>

```

ASPErrors

ASP3+

Syntax

Core ASP environment object

Description

The ASPErrors object is a core ASP environment object. Its properties and methods are used to retrieve information about errors that have occurred on a given page. This object is returned by the `Server.GetLastError()` method and exposes read-only properties. Table 9.2 contains a list of properties for this object.

Table 9.2 Properties of the ASPErrors Object

Property	Description
ASPCode	Returns the error code generated by the IIS Web server.
ASPDDescription	Returns a detailed description of an ASP-related error.
Category	Indicates the source of the error, which can be internal to ASP, the scripting language, or an object.
Column	Indicates the .asp file column position that generated the error.
Description	Returns a short description of the error.
File	Indicates the name of the .asp file being processed when the error occurred.
Line	Indicates the line that generated the error.
Number	Returns the standard COM error code.
Source	Returns the actual source code, when available, of the line that caused the error.

Example

Listing 9.13 shows how you can write out a short description of an error that has been exposed to the `ASPErrors` object.

Listing 9.13 Using the `ASPErrors` Object

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.Description);
</script>
```

ASPErrors.ASPCode

ASP3+

Syntax

`ASPErrors.ASPCode`

Description

The `ASPCode` property of the `ASPErrors` object contains error code generated by IIS.

Example

In Listing 9.14 we write out the value of the `ASPCode` property.

Listing 9.14 Using the `ASPCode` Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.ASPCode);
</script>
```

ASPErrors.ASPDescription

ASP3+

Syntax

`ASPErrors.ASPDescription`

Description

The `ASPDescription` property of the `ASPErrors` object contains a more detailed description of the error returned.

Example

In Listing 9.15 we write out the value of the `ASPDescription` property.

Listing 9.15 Using the `ASPDescription` Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.ASPDescription);
</script>
```

ASPError.Category

ASP3+

Syntax

ASPError.Category

Description

The Category property of the ASPError object indicates if the source of the error was internal to ASP, JScript, or an object.

Example

In Listing 9.16 we write out the value of the Category property.

Listing 9.16 Using the Category Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(ASPError.Category);  
</script>
```

ASPError.Column

ASP3+

Syntax

ASPError.Column

Description

The Column property of the ASPError object contains the column position within the ASP file where the error occurred.

Example

In Listing 9.17 we write out the value of the Column property.

Listing 9.17 Using the Column Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(ASPError.Column);  
</script>
```

ASPError.Description

ASP3+

Syntax

ASPError.Description

Description

The Description property of the ASPError object contains a short description of the error.

Example

In Listing 9.18 we write out the value of the Description property.

Listing 9.18 Using the Description Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(ASPError.Description);  
</script>
```

ASPError.File

ASP3+

Syntax

ASPError.File

Description

The File property of the ASPError object contains the name of the ASP file being processed when the error occurred.

Example

In Listing 9.19 we write out the value of the File property.

Listing 9.19 Using the File Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(ASPError.File);  
</script>
```

ASPError.Line

ASP3+

Syntax

ASPError.Line

Description

The Line property of the ASPError object contains the line number on which the error occurred.

Example

In Listing 9.20 we write out the value of the Line property.

Listing 9.20 Using the Line Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.Line);
</script>
```

ASPErrors.Number

ASP3+

Syntax

ASPErrors.Number

Description

The Number property of the ASPErrors object contains the standard COM error code.

Example

In Listing 9.21 we write out the value of the Number property.

Listing 9.21 Using the Number Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.Number);
</script>
```

ASPErrors.Source

ASP3+

Syntax

ASPErrors.Source

Description

The Source property of the ASPErrors object contains the actual source code, if available, that the error occurred in.

Example

In Listing 9.22 we write out the value of the Source property.

Listing 9.22 Using the Source Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(ASPErrors.Source);
</script>
```

blob() (Function)

ES2+

Syntax

`blob(path)`

Description

BLOB data represents *binary large objects* that can be stored in a database. This allows you to store various types of information, such as images, movie files, and sounds in the database.

NOTE

Be sure to consult the documentation on your specific database to see whether there are any limitations to BLOB data types.

The `blob()` function is used to store BLOB data in your database. This function takes the path to a BLOB file as its only parameter. Note that this path must be an absolute pathname and not a relative one.

Example

In Listing 9.23, a cursor instance has been created to perform a query on the database to find a specific row. Focus is then placed on that row and the `blob()` function is used to assign the data to a column. The final step in the process is to use the `updateRow()` method to commit the change.

Listing 9.23 Using the `blob()` Function to Insert BLOB Data into a Database

```
<server>

// SQL statement and instance of a cursor to execute it
var myStatement = 'SELECT * FROM family WHERE pic = null';
var myCursor = database.cursor(myStatement);

// Iterate through the returned rows.
while(myCursor.next()){

    // Assign 'blank.gif' in the PIC column of the returned rows.
    myCursor.pic = blob("/pictures/blank.gif");
    myCursor.updateRow("family");
}

// Close the cursor and write it to the page if there was an error.
var dbError = myCursor.close();

if(dbError) write(myCursor.close());
</server>
```

blob (Object)

ES2+

Syntax

Core object is created with the `blob.blobImage()` and `blob.blobLink()` methods.

Description

The top-level `blob` object contains methods that allow you to store and retrieve blob data in a database. *BLOB* data represents *binary large objects* that can be stored in a database. This allows you to store various types of information, such as images, movie files, and sounds in the database.

NOTE

Be sure to consult the documentation on your specific database to see whether there are any limitations to BLOB data types.

The core `blob` object is created when you use the methods of this object. Table 9.3 has the methods of the `blob` object and a description of what they do when invoked.

Table 9.3 *Methods of the blob Object*

Method	Description
<code>blobImage()</code>	Retrieves and displays a BLOB data instance stored in a database
<code>blobLink()</code>	Retrieves and displays a link that references a BLOB data instance stored in a database
<code>unwatch()</code>	Turns off the watch for a particular property.
<code>watch()</code>	Turns on the watch for a particular property.

Example

Listing 9.24 queries a database for a specific image, creating an instance of a `blob` object when retrieved. The image is then written to the page using the `write()` and `blobImage()` methods. The actual tag written will be as follows:

```
<IMG ALT="Click" ALIGN="left" WIDTH="468" HEIGHT="60" BORDER="0" ISMAP="false">
```

Listing 9.24 Using the `blobImage()` Method to Format an `<img>` Tag

```
<server>
```

```
// Find the image you want to display.
myCursor = myConn.cursor("SELECT path FROM images WHERE img = 1");

// Write the <img> tag to the page with the following attributes set.
write(myCursor.path.blobImage("gif", "Click", "left", 468, 60, 0, false));
```

Listing 9.24 Continued

```
// Close the cursor.  
myCursor.close();  
</server>
```

blob.blobImage()

ES2+

Syntax

`cursor.column.blobImage(fileType, altText, align, width, height, border, ismap)`

Description

The `blobImage()` method retrieves and displays a BLOB image stored in a database. The method actually returns the HTML for the `<img>` tag used to display the image. The `href` attribute of the `<img>` tag references the instance of this image in memory and does not have to contain a “normal” URL of the image itself.

This method can take up to seven parameters that set the various attributes of the `<img>` tag. These attributes are contained in Table 9.4. At a minimum, you must pass the *fileType* of the image.

Table 9.4 Properties That Can Be Set with the `blobImage()` Method

Parameter	Attribute It Sets	Description
<code>fileType</code>	<code>none</code>	This parameter does not set an attribute. It specifies the type of file that is being displayed, such as <code>gif</code> or <code>jpeg</code> .
<code>altText</code>	<code>alt</code>	A string that is displayed when the browser has been set to not display images or when a mouse is over an image for a specified period of time.
<code>align</code>	<code>align</code>	This can be set to <code>LEFT</code> , <code>RIGHT</code> , or any other value your target browser supports.
<code>width</code>	<code>width</code>	The width in pixels of your image.
<code>height</code>	<code>height</code>	The height in pixels of your image.
<code>border</code>	<code>border</code>	An integer value that specifies the size of any border that might appear around the image if the image is surrounded by the <code><a></code> tag.
<code>ismap</code>	<code>ismap</code>	Specifies whether the image has a map file associated with it to handle any clicks that might occur within it. This parameter is set by specifying <code>true</code> if the image has a map file associated with it or <code>false</code> if not.

Example

Listing 9.25 queries a database for a specific image. The image is then written to the page using the `write()` and `blobImage()` methods. The actual tag written will be as follows:

```
<IMG ALT="Click" ALIGN="left" WIDTH="468" HEIGHT="60" BORDER="0"
  ↳ISMAP="false">
```

Listing 9.25 *Using the `blobImage()` Method to Format an `<img>` Tag*

```
<server>

// Find the image you want to display.
myCursor = myConn.cursor("SELECT path FROM images WHERE img = 1");

// Write the <img> tag to the page with the following attributes set.
write(myCursor.path.blobImage("gif", "Click", "left", 468, 60, 0, false));

// Close the cursor.
myCursor.close();
</server>
```

blob.blobLink()

ES2+

Syntax

```
cursor.column.blobLink(mimeType, text)
```

Description

The `blobLink()` method retrieves BLOB data stored in a database, stores it in memory, and creates a temporary link to it. The method actually returns the HTML for the `<a>` tag used to display the link. The `href` attribute of the `<a>` tag references the BLOB data type, which has been stored in a temporary memory location, and does not contain a “normal” URL of this attribute. The data is stored in memory until the user clicks the link or until 60 seconds have elapsed.

The parameters this method takes are the MIME type of the file referenced, and the `text` that is displayed to the user as a link.

Example

Listing 9.26 queries a database for a specific image. A link referencing the image is then written to the page using the `write()` and `blobLink()` methods. The actual tag written will be as follows:

```
<a href="LIVEWIRE_TEMP1">Click Here!</a>
```

Listing 9.26 Using the `blobLink()` Method to Format an `<a>` Tag

```
<server>

// Find the image you want to display.
myCursor = myConn.cursor("SELECT path FROM images WHERE img = 1");

// Write the <a> tag to the page with the attributes set.
write(myCursor.path.blobLink("image/gif", "Click Here!"));

// Close the cursor.
myCursor.close();
</server>
```

callC()

ES2+

Syntax

```
callC(JSFuncName, arg1, arg2, ..., argN)
```

Description

The `callC()` function, which returns string values, is a top-level function that is not associated with any core object. `callC()` is used to call a JavaScript function that references a C function in a shared library. These libraries are the pre-built `.dll` files on Windows machine and `.so` files on Unix machines. `callC()` takes the JavaScript name you have assigned to the C function and any arguments the function needs as arguments.

Before you can call this function, you must register the C library using the server-side JavaScript `registerCFunction()`. `registerCFunction()` takes the JavaScript name with which you want to reference the function, the path to the library, and the C function name as parameters.

Example

Listing 9.27 registers an external C library, `extlib.dll`, that contains a function named `getMyDate`. The registration of this function assigns the name `JSExtLib` to be used within the script. If the function successfully registers, the `callC()` function is used to call the C function and pass it two parameters. The results are written to the user's page. If the function does not register properly, an error is written to the user's page.

Listing 9.27 Using a C Function with `callC` That Has Been Registered

```
<server>

// Register the library and function, assigning it a JavaScript
// function name.
```

```

var myExternalLib = registerCFunction("JSExtLib", "c:/winnt/extlib.dll",
  ↪"getMyDate")

// If the library registered without error, then call it using the
// callC function. If it failed, then write an error to the page.
if (myExternalLib) {
  write(callC("getMyDate", 1999, 2000));
}else{
  write("There was an error processing this external library function");
}

</server>

```

client

ES2+

Syntax

Core object is created with each connection of a client to your application.

Description

An instance of the `client` object is created with each connection of a user to your application. This object is used to maintain session variables for that user as she moves through your application's pages. Because the object is not created until a user connects to your application, you cannot use the object on the first page of your application.

NOTE

The `client` object is created for each user's connection to each application you have built using Server-side JavaScript, so a single user connected to two applications will have two `client` objects created.

The object itself is held until the user is inactive for a set period of time or the object is destroyed. At that time, the JavaScript runtime engine cleans up the object. The default timeout is ten minutes, but this can be changed by using the `expiration()` method or can be destroyed manually by using the `destroy()` method.

`client` objects do not have any default properties, but properties can be created for them. Do note that because of the method used to maintain user sessions, these properties are all converted to strings. If you have created a property that must be evaluated as a numeric value, use the `parseInt()` and `parseFloat()` methods for processing.

If you must store an object as a property, you will have to create an array of objects in the `project` or `server` objects. Then you can create a property to hold the index of your object in the `client` object.

Example

Listing 9.28 contains a form the user fills out with her name, e-mail address, and phone number. When the Submit button is clicked, the form is sent back to itself and the script sees information being passed in. This information is then assigned to three created properties of the client object and is then written to the user's page.

Listing 9.28 *Assigning Properties to the client Object*

```
<html>
<head>
  <title>Using the client object</title>
</head>
<body>
<server>

// See if they have submitted or just need the form.
if(request.method == "POST"){

  // Assign the client properties their values.
  client.name = request.name;
  client.email = request.email;
  client.phone = request.phone;

  // Write the user's information to the page.
  write('Hello ' + client.name + '!<br>');
  write('Please confirm your email, ' + client.email + ', and ');
  write('phone number, ' + client.phone);
}else{

  // If this page was called and a form was not submitted
  write('<form name="myForm" method="post">');
  write('<table border="1"><tr><td>');
  write('<table border="0">');
  write('<tr align="left" valign="top">');
  write('<td><b>Name:</b></td>');
  write('<td><input type="text" name="name" size="30"></td>');
  write('</tr>');
  write('<tr align="left" valign="top">');
  write('<td><b>E-mail:</b></td>');
  write('<td><input type="text" name="email" size="30"></td>');
  write('</tr>');
  write('<tr align="left" valign="top">');
  write('<td><b>Phone:</b></td>');
  write('<td><input type="text" name="phone" size="30"></td>');
  write('</tr>');
  write('<tr align="left" valign="top">');
  write('<td colspan="2" align="right"><input type="submit">');
  write(' value="Submit"></td>');
  write('</tr>');
```



```
    write('</table>');  
    write('</td></tr></table>');  
    write('</form>');  
}  
</server>  
</body>  
</html>
```

client.destroy()

ES2+

Syntax

```
client.destroy()
```

Description

The `destroy()` method of the `client` object explicitly destroys that instance of the object and all its associated properties. If this method is not called, the JavaScript runtime will destroy the object after 10 minutes or after the time specified with the `client.expiration()` method.

If you are using cookies to maintain your `client` object, calling the `destroy()` method acts in the same manner but does not remove information stored in the browser's cookie file. To remove the cookie information, set the expiration to 0 seconds by using the `client.expiration()` method.

When using URL encoding to maintain the `client` object, the `destroy()` method will destroy all information with the exception that the links created before the method call will retain their properties. Because of this, good programming practice warrants calling the method at the top or bottom of a page.

Example

Listing 9.29 shows how to destroy the properties of your user's `client` object.

Listing 9.29 Using the `destroy()` Method to Destroy the `client` Object Properties

```
<server>  
  
// Destroy the client properties.  
client.destroy()  
  
</server>
```

client.expiration()

ES2+

Syntax

```
client.expiration(seconds)
```

Description

The `expiration()` method of the `client` object sets the number of *seconds* of user inactivity before the JavaScript runtime destroys all properties associated with that session. The default timeout is 10 minutes if you do not explicitly set this property. Also, this method has no effect when using URL encoding to maintain your `client` objects.

NOTE

Setting this to 0 will remove any cookies associated with the `client` object when using client cookies to maintain sessions.

Example

Listing 9.30 sets the destruction of the `client` object to occur after five minutes of inactivity.

Listing 9.30 Using the `expiration()` Method of the `client` Object

```
<server>
```

```
// Set the expiration to 5 minutes.  
client.expiration(300)
```

```
</server>
```

client.property

ES2+

Syntax

```
client.property
```

Description

When a *property* is created for an instance of the `client` object, it is accomplished by passing information from a form. When you pass a form element, designated by the `name` attribute, to the server, it is accessible through the `client.property` syntax, where *property* is the name passed.

Example

Listing 9.31 shows how the form value `zip` is accessible in a server-side JavaScript script by assigning it to an instance of the `client` object. Once assigned, the value stored in the property is written back out to the user's page.

Listing 9.31 Accessing a Client Property

```

<server>

// See if they have submitted.
if(request.method == "POST"){

    // Assign the client property its value.
    client.zip = request.zip;
}

// Write the value passed by the form to the page.
write(client.zip);
</server>

```

client.unwatch()

ES3+

Syntax

```
client.unwatch(property)
```

Description

The `unwatch()` method of the `client` object is used to turn off the watch for a particular *property*.

Example

Listing 9.32 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.32 Example of the `unwatch()` Method

```

<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// See if they have submitted.
if(request.method == "POST"){

    // Assign the client property its value.
    client.zip = request.zip;
}

// watch property
client.watch("zip", alertme);

```

Listing 9.32 Continued

```
// change value
client.zip = null;

// turn off watch
client.unwatch("zip");

// change value again
client.zip = 3;

</server>
```

client.watch()

ES3+

Syntax

```
client.watch(property, function)
```

Description

The `watch()` method of the `client` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.33 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.33 Example of the watch() Method

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// See if they have submitted.
if(request.method == "POST"){

    // Assign the client property its value.
    client.zip = request.zip;
}

// watch property
client.watch("zip", alertme);
```

```
client.zip = null;
```

```
</server>
```

Connection

ES3+

Syntax

Core object is created when the `DbPool1.connection()` method is called.

Description

The `Connection` object represents a given connection, pulled from a “pool”, to a database. This object has only one property, the `prototype` property, which you can use to add properties to the object. Table 9.5 shows the methods associated with this object.

NOTE

If you only need a single connection to the database and do not need to create a pool, use the `database` object for your connection.

Table 9.5 *Methods of the Connection Object*

Method	Description
<code>beginTransaction()</code>	Begins a new SQL transaction
<code>commitTransaction()</code>	Commits the current SQL transaction
<code>connected()</code>	Tests to see whether the pool connection is connected to the database
<code>cursor()</code>	Creates a <code>cursor</code> object for the specified SQL <code>SELECT</code> statement
<code>execute()</code>	Performs the non- <code>SELECT</code> SQL statement passed
<code>majorErrorCode()</code>	Returns the major error code numeric value returned by the database or ODBC
<code>majorErrorMessage()</code>	Returns the major error message string value returned by the database or ODBC
<code>minorErrorCode()</code>	Returns the secondary error code numeric value returned by the database or ODBC
<code>minorErrorMessage()</code>	Returns the secondary error message string value returned by the database or ODBC
<code>release()</code>	Releases the specified connection back to the pool
<code>rollbackTransaction()</code>	Rolls back the specified transaction
<code>SQLTable()</code>	Formats the query results from a <code>SELECT</code> in HTML <code><table></code> format for easy writing to a client

Table 9.5 *Continued*

Method	Description
<code>storedProc()</code>	Creates a <code>StProc</code> object and runs the specified stored procedure
<code>toString()</code>	Returns a string representing the specified object
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property

Example

Listing 9.34 creates a pool of connections to an Oracle database and initializes a connection from that pool. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates the user's name. If a connection is not made, the error code and message is returned to the screen.

Listing 9.34 *Creating and Using a Connection Object*

```
<server>
```

```
// Assign the user submitted ID and name to the client object as properties.
```

```
client.uid = request.uid;
```

```
client.name = request.name;
```

```
// Create a pool of connections.
```

```
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
// Open a connection from the pool. Give error if connection could
```

```
// not be made.
```

```
var myConn = myPool.connection('Employees', 15);
```

```
if(myConn.connected()){
```

```
    // Start a new SQL transaction to perform a SELECT.
```

```
    myConn.beginTransaction();
```

```
    var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = ' +
```

```
    ➤ client.uid);
```

```
    // Focus on that line, change the name column for that user,
```

```
    // and update the row.
```

```
    currRow.next();
```

```
    currRow.name = client.name;
```

```
    currRow.updateRow("employees");
```

```
    // Close the cursor.
```

```
    currRow.close();
```

```
// If the connection fails, write an error message.
```

```
}else{
```

```
write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}
```

```
</server>
```

Connection.beginTransaction()

ES3+

Syntax

```
connection.beginTransaction()
```

Description

The `beginTransaction()` method of the `Connection` object begins a new SQL transaction. This groups all the actions against the database together until the user exits the page or either the `commitTransaction()` or `rollbackTransaction()` methods are called. In the instance of the user exiting the page, the transaction is either committed or rolled back, depending on the setting of the commit flag when the `DbPool` object instance is created.

NOTE

You cannot have nested transactions.

Example

Listing 9.35 creates a pool of connections to an Oracle database and pulls one of the connections from the pool. After the connection has been verified, the `beginTransaction()` method is called and a SQL query is performed. The results are formatted in a table with the `SQLTable()` method and written to the user's page.

Listing 9.35 *Starting a New Transaction with the* `beginTransaction()` *Method*

```
<server>
```

```
// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){
```

Listing 9.35 Continued

```
// Start a new transaction and write the results to a page, formatting
// them with the SQLTable method.
myConn.beginTransaction();
write(myConn.SQLTable('SELECT * FROM employees WHERE uid >= '+client.uid));

// Commit the transaction.
myConn.commitTransaction();

// If the connection fails, write an error message
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Connection.commitTransaction()

ES3+

Syntax

```
connection.commitTransaction()
```

Description

The `commitTransaction()` method of the `Connection` object commits a new SQL transaction. This commits all the actions against the database since the last commit. If the commit is successful, `0` is returned. If a non-zero number is returned, an error is encountered. In this case, various methods of the `Connection` object can be used to retrieve the code and message of the error.

Example

Listing 9.36 creates a pool of connections to an Oracle database and pulls one of the connections from the pool. After the connection has been verified, the `beginTransaction()` method is called and a SQL query is performed. The results are formatted in a table with the `SQLTable()` method and written to the user's page.

Listing 9.36 Starting a New Transaction with the `commitTransaction()` Method

```
<server>

// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```



```
// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new transaction and write the results to a page, formatting
    // them with the SQLTable method.
    myConn.beginTransaction();
    write(myConn.SQLTable('SELECT * FROM employees WHERE uid >= '+client.uid));

    // Commit the transaction.
    myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ( ' + myConn.majorErrorCode()+'): ' +
myConn.majorErrorMessage());
}

</server>
```

Connection.connected()

ES3+

Syntax

```
connection.connected()
```

Description

The `connected()` method of the `Connection` object tells if the pool of connections to the database is still connected.

Example

Listing 9.37 creates a pool of connections and pulls a connection from the pool for processing. If the connection is made, any code within that section is executed. If the connection fails, the error is written to the page.

Listing 9.37 Testing a Connection with the `connected()` Method

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
```

Listing 9.37 Continued

```
if (myConn.connected()) {  
  
    // You are connected, so perform any tasks here.  
  
}else{  
  
    // There was an error connecting to the database.  
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());  
}  
  
</server>
```

Connection.cursor()

ES3+

Syntax

```
connection.cursor(sql)  
connection.cursor(sql, boolean)
```

Description

The `cursor()` method of the `Connection` object creates a `Cursor` object that can be used to run SQL queries against the database. The method takes the *sql* statement as a parameter, as well as an optional *boolean* value that specifies whether the cursor is updateable.

Example

Listing 9.38 shows how you would run a query against the database using the `cursor()` method. The while loop is used to write the results to the user's page.

Listing 9.38 Using the `cursor()` Method to Run a Query Against the Database

```
<server>  
  
// Set the query to run.  
var mySQL = myConn.cursor('SELECT name,title FROM employees');  
  
// Iterate through the results and write them to the page.  
while(mySQL.next()){  
    write(mySQL.name + ': ' + mySQL.title + '<br>');  
}  
  
</server>
```

Connection.execute()

ES3+

Syntax

```
connection.execute(statement)
```

Description

The `execute()` method of the `Connection` object enables your application to execute a *DDL (Data Definition Language)* or *DML (Data Manipulation Language)* query, which does not return a `Cursor`, supported by your database. This includes statements such as `CREATE`, `ALTER`, and `DROP`.



NOTE

Be sure to use SQL that conforms to your database.

Example

Listing 9.39 deletes all rows with a UID less than the number passed to the script.

Listing 9.39 Using the `execute()` Method to Run DML Queries

```
<server>
```

```
// Assign the UID passed to the client object
client.uid = request.uid;
```

```
// Execute a DELETE based on the UID passed
myConn.execute('DELETE FROM employees WHERE uid < ' + client.uid);
```

```
</server>
```

Connection.majorErrorCode()

ES3+

Syntax

```
connection.majorErrorCode()
```

Description

The `majorErrorMessage()` method of the `Connection` object contains the ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.40 shows how you would create a pool of connections, pull a connection from it, and test for the connection. If the test fails, the `majorErrorCode()` is used when writing the error to the page.

Listing 9.40 *Using `majorErrorCode()` to Retrieve a Database Connection Error*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Connection.majorErrorMessage()

ES3+

Syntax

```
connection.majorErrorMessage()
```

Description

The `majorErrorMessage()` method of the `Connection` object contains the ODBC or database string error message that is returned if an error occurs.

Example

Listing 9.41 shows how you would create a pool of connections, pull a connection from it, and test for the connection. If the test fails, the `majorErrorMessage()` is used when writing the error to the page.

Listing 9.41 *Using `majorErrorMessage()` to Retrieve a Database Connection Error*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
```

```
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Connection.minorErrorCode()

ES3+

Syntax

```
connection.minorErrorCode()
```

Description

The `minorErrorMessage()` method of the `Connection` object contains the secondary ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.42 shows how you would create a pool of connections, pull a connection from it, and test for the connection. If the test fails, the `minorErrorCode()` is used when writing the secondary error to the page.

Listing 9.42 *Using `minorErrorCode()` to Retrieve a Secondary Database Connection Error*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.
```

Listing 9.42 Continued

```
}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.minorErrorCode()+'): '+myConn.minorErrorMessage);
}

</server>
```

Connection.minorErrorMessage()

ES3+

Syntax

```
connection.minorErrorMessage()
```

Description

The `minorErrorMessage()` method of the `Connection` object contains the secondary ODBC or the database string error message that is returned if an error occurs.

Example

Listing 9.43 shows how you would create a pool of connections, pull a connection from it, and test for the connection. If the test fails, the `minorErrorMessage()` is used when writing the secondary error to the page.

Listing 9.43 Using `minorErrorMessage()` to Retrieve a Secondary Database Connection Error

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.minorErrorCode()+'): '+myConn.minorErrorMessage);
}
```

Connection.prototype

ES3+

Syntax

```
connection.prototype.method = name
connection.prototype.property = value
```

Description

The prototype property of the Connection object allows you to add methods and properties to the Connection object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.44 creates a new property and method of the Connection object. An instance is created and the new property is set. The new method is then called to verify the property and, if it is incorrect (which it is), an error message is written to the page.

Listing 9.44 *Using the prototype Property to Create a New Property and Method*

```
<server>
```

```
// Define the method that we prototyped.
function verifyODBC(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "ODBC"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the Connection object.
Connection.prototype.type = null;
Connection.prototype.isODBC = verifyODBC;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool.
var myConn = myPool.connection('Employees', 15);

// Using the prototype we defined, assign the type property.
myConn.type = "Oracle";
```

Listing 9.44 Continued

```
// Check the type of the connection to see if it is valid.
if(myConn.isODBC()){
    write(myConn + " has a valid type of " + myConn.type);
}else{
    write(myConn + " has an invalid type of " + myConn.type);
}

</server>
```

Connection.release()

ES3+

Syntax

```
connection.release()
```

Description

The `release()` method of the `Connection` object returns the connection to the `DbPool` instance after all cursors have been closed. If you do not close the cursor, the connection will remain until it times out or the variable holding your connection, assuming you assigned it to one, goes out of scope. Depending on how you have written your application, this can happen when the application is stopped, the Web server is stopped, or when control leaves the HTML page.

Example

Listing 9.45 shows a pool being created, a connection being pulled from the pool, and a query run against the database. When the cursor is closed, the connection is released.

Listing 9.45 Releasing a Connection Back to the Pool Using the `release()` Method

```
<server>

// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
```



```

var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = ' +
↳ client.uid);

// Focus on that line, change the name column for that user,
// and update the row.
currRow.next();
currRow.name = client.name;
currRow.updateRow("employees");

// Close the cursor.
currRow.close();

// If the connection fails, write an error message.
}else{
  write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>

```

Connection.rollbackTransaction()

ES3+

Syntax

```
connection.rollbackTransaction()
```

Description

The `rollbackTransaction()` method of the `Connection` object will undo all actions performed since the last `beginTransaction()` method call.

NOTE

You cannot have nested transactions.

Example

Listing 9.46 takes a *commit* field sent to the application from the user. If this evaluates to true, the transaction is committed. If not, it is rolled back.

Listing 9.46 Rolling Back a Transaction with the `rollbackTransaction()` Method

```

<server>

// See if the user wants to commit the last transaction.
client.commit = request.commit;

```

Listing 9.46 Continued

```
if(client.commit = "YES"){  
  
    // Commit the transaction.  
    myConn.commitTransaction();  
  
}else{  
  
    // Rollback the transaction.  
    myConn.rollbackTransaction();  
}  
  
</server>
```

Connection.SQLTable()

ES3+

Syntax

```
connection.SQLTable(sql)
```

Description

The `SQLTable()` method of the `Connection` object takes a `sql` `SELECT` statement as a parameter and executes a query through the connection from which it was called. It returns the result formatted in an HTML table for easy writing to a client's page. This is a simple table in the following format:

```
<table border="1">  
<tr>  
<th>column 1</th>  
<th>column 2</th>  
...  
<th>column N</th>  
</tr>  
<tr>  
<td>value 1 of column 1</td>  
<td>value 1 of column 2</td>  
...  
<td>value 1 of column N</td>  
</tr>  
<tr>  
<td>value 2 of column 1</td>  
<td>value 2 of column 2</td>  
...  
<td>value 2 of column N</td>  
</tr>  
...  
</table>
```

Example

Listing 9.47 runs a user passed query and formats the result using the `SQLTable()` method. This information is then written to the user's page.

Listing 9.47 *Using the `SQLTable()` Method to Format the Results of a SELECT Query*

```
<server>

// Assign the user submitted query to the client object.
client.sql = request.sql;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new transaction and write the results to a page, formatting
    // them with the SQLTable method.
    myConn.beginTransaction();
    write(myConn.SQLTable(client.sql));

    // Commit the transaction.
    myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>
```

Connection.storedProc()

ES3+

Syntax

```
connection.storedProc(procName)
connection.storedProc(procName, arg1, arg2, ... , argN)
```

Description

The `storedProc()` method of the `Connection` object creates a `Stproc` object that allows you to execute a database-specific stored procedure using the connection from which it was invoked.

As shown in the syntactical definition, you can also pass any arguments needed to the method for processing. If you are using a stored procedure that requires arguments, or if you want to have the procedure run using default arguments, you must pass `/Default/` as the argument. The following shows an example of passing a default value:

```
var myStproc = myConn.storedProc("sp_employees", "/Default/");
```

The scope of this procedure is restricted to the current page. Any methods of the `Stproc` object must be invoked on the current page. If this is not possible, a new object will have to be created on subsequent pages to access the properties needed.

Example

Listing 9.48 creates a pool of connections and pulls one of the connections. When the connection has been verified, the `storedProc()` method is used to invoke the fictitious `sp_employees` stored procedure.

Listing 9.48 Using the `storedProc()` Method to Invoke a Stored Procedure on a Database

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    myConn.beginTransaction();

    // Run the stored procedure.
    var myStproc = myConn.storedProc("sp_employees");

    // Commit the transaction.
    myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}
```

```
// Release the connection.
myConn.release();
```

```
</server> >
```

Connection.toString()

ES3+

Syntax

```
connection.toString()
```

Description

The `toString()` method of the `Connection` object returns a text value of the object. When invoked on an instance of a `Connection` object, the string is returned in the following format:

```
"dbName" "uid" "dbType" "dbInstance"
```

If the parameter is unknown, an empty string is returned. Table 9.6 contains the value of these returned values.

Table 9.6 *Return Values of the `toString()` Method*

Method	Description
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections, this is a blank ("") string. In Oracle, the name of the database for these connections is set up in the <code>tnsnames.ora</code> file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the <code>dbInstance</code> .
uid	The username or ID you want the connections to connect as.
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC, it is the DSN entry name.

Example

Listing 9.49 creates an instance of the `Connection` object. Once created, the `write()` method is used to write its string value to the page.

Listing 9.49 Write the Results of Calling the toString() Method to a Page

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Write the string value of the object to the page.
    write(myConn.toString());

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>
```

Connection.unwatch()

ES3+

Syntax

```
connection.unwatch(property)
```

Description

The `unwatch()` method of the `Connection` object is used to turn off the watch for a particular *property*.

Example

Listing 9.50 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.50 Example of the unwatch() Method

```
<server>

// Define the method that we prototyped.
function verifyODBC(){
```

```
// Check to see if the type property we added is set to a valid value.
if(this.type == "ODBC"){
    return true;
}else{
    return false;
}
}

// Function that is called if property's value changes.
function alertme(id, oldValue, newValue){
    write("ID ( " + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Connection object.
Connection.prototype.type = null;
Connection.prototype.isODBC = verifyODBC;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool.
var myConn = myPool.connection('Employees', 15);

// Using the prototype we defined, assign the type property.
myConn.type = "Oracle";

// Watch the type property.
myConn.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myConn.isODBC()){
    write(myConn + " has a valid type of " + myConn.type);
}else{
    write(myConn + " has an invalid type of " + myConn.type);
}

// change value
myConn.type = null;

// turn off watch
myConn.unwatch("type");

// change value again
myConn.type = "Sybase";

</server>
```

Connection.watch()

ES3+

Syntax

```
connection.watch(property, function)
```

Description

The `watch()` method of the `Connection` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.51 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.51 Example of the `watch()` Method

```
<server>
```

```
// Define the method that we prototyped.
function verifyODBC(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "ODBC"){
        return true;
    }else{
        return false;
    }
}

// Function that is called if property's value changes.
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Connection object.
Connection.prototype.type = null;
Connection.prototype.isODBC = verifyODBC;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool.
var myConn = myPool.connection('Employees', 15);

// Using the prototype we defined, assign the type property.
myConn.type = "Oracle";
```



```
// Watch the type property.
myConn.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myConn.isODBC()){
    write(myConn + " has a valid type of " + myConn.type);
}else{
    write(myConn + " has an invalid type of " + myConn.type);
}

// change value
myConn.type = null;

</server>
```

Cursor

ES2+

Syntax

```
connection.cursor()
database.cursor()
```

Description

The Cursor object is a core object created when the `cursor()` method of the `Connection` or `database` object is called. A database query is said to return a cursor, so this object contains references to the rows returned from a query.

When working with cursor objects, you should explicitly close them using the `close()` method when you are finished. Not doing so will cause the JavaScript runtime to hold the cursor in memory until the connection or pool to which the cursor was tied goes out of scope.

The Cursor object has several methods and properties associated with it. These are listed in Table 9.7.

Table 9.7 *Properties and Methods of the Cursor Object*

Type	Item	Description
property	<i>columnName</i>	This property represents the column names that are returned from the SQL statement you passed to the <code>cursor()</code> method.
	<i>prototype</i>	This property allows you to add methods and properties to the <code>cursor</code> object to be used when new instances are created.
method	<code>close()</code>	This method closes the cursor and frees any memory used by it.

Table 9.7 Continued

Type	Item	Description
	<code>columnName()</code>	This method takes an indexed numbered location and returns the column name of the column in that location.
	<code>columns()</code>	This method returns the number of columns in the cursor.
	<code>deleteRow()</code>	This method deletes the current row of the table passed to the method.
	<code>insertRow()</code>	This method inserts a new row in the table passed to the method.
	<code>next()</code>	This method moves from the current row in the cursor object to the next row.
	<code>unwatch()</code>	This method is used to turn off the watch for a particular property.
	<code>updateRow()</code>	This method updates the current row in the table passed to the method.
	<code>watch()</code>	This method is used to turn on the watch for a particular property.

Example

Listing 9.52 takes a UID, passed as an area code, that is assigned to the `client` object. A pool of connections is then opened to the database, and one of the connections is pulled to run the query. The results of the query are stored in a `Cursor` object and are iterated through use of the `next()` method. After all rows have been updated with the new area code, the cursor is closed and the connection is released.

Listing 9.52 Using the Cursor Object

```
<server>

// Assign the user submitted ID and area code to the client object
// as properties.
client.uid = request.uid;
client.areacode = request.areacode;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
```

```

    var currRow = myConn.cursor('SELECT areacode FROM employees WHERE uid >= '
➤ + client.uid);

    // For all the lines that matched, update the area code.
    while(currRow.next()){
        currRow.areacode = client.areacode;
        currRow.updateRow("employees");
    }

    // Close the cursor.
    currRow.close();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>

```

Cursor.close()

ES2+

Syntax

```
cursor.close()
```

Description

The `close()` method of the `Cursor` object closes the cursor and frees all memory that had been used to store its information. If successful, the method returns `0`, otherwise it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or database objects.

Example

Listing 9.53 creates a cursor and then closes it.

Listing 9.53 Closing a Cursor with the `close()` Method

```

<server>

// Create cursor
var currRow = myConn.cursor('SELECT areacode FROM employees WHERE uid >= '
➤ + client.uid);

```

Listing 9.53 Continued

```
// Close the cursor
currRow.close();

</server>
```

Cursor.columnName

ES2+

Syntax*cursor.columnName***Description**

The *columnName* property of the *Cursor* object is an array of objects that corresponds to the name of the columns in the cursor.

Example

Listing 9.54 shows a cursor object being created. The various values are then written to the page using the *columnName* property as their reference.

Listing 9.54 Using the columnName Property

```
<server>

var currRow = myConn.cursor('SELECT areacode,phone,name FROM employees');

// Write each person's name and phone number to the page in the form:
// "<name>'s phone number is (<areacode>) <phone>"
while(currRow.next()){
    write(currRow.name + "'s phone number is (" + currRow.areacode + ") ");
    write(currRow.phone + "<br>");
}

// Close the cursor
currRow.close();

</server>
```

Cursor.columnName()

ES2+

Syntax*cursor.columnName(num)*

Description

The `columnName()` method of the `Cursor` object takes the zero-based indexed number location, *num*, passed to the method and returns the name of the column in that location. Note that these names are not returned in any specific order unless you order them as such. Successive calls to the method, however, will return all the columns. See the example for more information on this.

Example

Listing 9.55 has two cursors. One of the cursors returns specific column names and the other returns all columns. See the comments in the code for the output.

Listing 9.55 *Using the `columnName()` Method to Return the Names of the Columns in a Table*

<server>

```
// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
    var currRow1 = myConn.cursor('SELECT areacode,phone FROM employees
    ↪WHERE uid >= 100');
    var currRow2 = myConn.cursor('SELECT * FROM employees WHERE uid >= 100');

    // Writes 'areacode', from the first cursor, to the page.
    write(currRow1.columnName(0));

    // Writes 'phone', from the first cursor, to the page.
    write(currRow1.columnName(1));

    // Writes all column names stored in the second cursor to the page.
    // This will include 'areacode' and 'phone' as well as any other
    // columns.
    for(var i = 0; i <= currRow2.columns(); i++){
        write(currRow2.columnName(i));
    }

    // Close the cursors.
    currRow1.close();
    currRow2.close();
}
```

Listing 9.55 Continued

```
// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>
```

Cursor.columns()

ES2+

Syntax

```
cursor.columns()
```

Description

The `columns()` method of the `Cursor` object returns the number of columns in the cursor on which it is invoked. If the SQL string that was passed to create the cursor specified a set number of columns to return, this is the number returned by the method.

Example

Listing 9.56 shows how you can return all the column names of the columns in your cursor.

Listing 9.56 Using the `columns()` Method to Determine How Many Columns Are in the Cursor Before Writing Them to the Page

```
<server>

var currRow = myConn.cursor('SELECT * FROM employees');

// Writes all column names stored in the cursor to the page.
for(var i = 0; i <= currRow.columns(); i++){
    write(currRow.columnName(i));
}

// Close the cursors
currRow.close();

</server>
```

Cursor.deleteRow()

ES2+

Syntax

```
cursor.deleteRow(table)
```

Description

The `deleteRow()` method of the `Cursor` object uses an updateable cursor and deletes the current row in the specified *table* of the `Cursor` object. If the delete was successful, `0` is returned; otherwise it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or `database` objects.

Example

Listing 9.57 creates a `Cursor` object and selects all instances of a given *id*. The `deleteRow()` method is then used to delete each of these instances.

Listing 9.57 *Deleting a Row from the Cursor Using the deleteRow() Method*

```
<server>
```

```
// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = '
➤ + client.uid, true);

    // Delete each row in the cursor.
    while(currRow.next()){
        currRow.delete("employees");
    }

    // Close the cursor.
    currRow.close();
```

Listing 9.57 Continued

```
// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Cursor.insertRow()

ES2+

Syntax

```
cursor.insertRow(table)
```

Description

The `insertRow()` method of the `Cursor` object uses an updateable cursor and inserts a new row in the specified *table* of the `Cursor` object. If the insert was successful, `0` is returned; otherwise, it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or `data-base` objects.



NOTE

Depending on your database, you might have to close the current cursor and reopen it if you want to access a newly inserted row. Also, if the `next()` method has been called on the cursor, any columns you do not specify values for will get the same values as the current row.

Example

Listing 9.58 creates a `Cursor` object and selects all rows. The `insertRow()` method is then used to insert a new row with three specified columns.

Listing 9.58 Inserting a Row from the Cursor Using the `insertRow()` Method

```
<server>

// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;
client.name = request.name;
client.pwd = request.pwd;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```



```
// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT. Notice the
    // cursor is updateable.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT uid,name,pwd FROM employees', true);

    // Assign values to the columns and insert a new row.
    currRow.uid = client.uid;
    currRow.name = client.name;
    currRow.pwd = client.pwd;
    currRow.insertRow("employees");
    myConn.commitTransaction();

    // Close the cursor and release the connection.
    currRow.close();
    myConn.release();

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Cursor.next()

ES2+

Syntax

cursor.next()

Description

The next() method of the Cursor object moves the point in the current row to the next row in the cursor. This method is used to iterate through each of the rows returned by the cursor. This method returns true, unless it is the last row of the cursor, at which time it returns false.

Example

Listing 9.59 creates an instance of the Cursor object and iterates through its results, deleting each row. This is performed by using the next() method.

Listing 9.59 Using the `next()` Method to Iterate Through the Rows in a Cursor

```
<server>

var currRow = myConn.cursor('SELECT * FROM employees WHERE uid <= 200');

// Delete each row in the cursor.
while(currRow.next()){
    currRow.delete("employees");
}

// Close the cursor.
currRow.close();

</server>
```

Cursor.prototype

ES2+

Syntax

```
cursor.prototype.method = name
cursor.prototype.property = value
```

Description

The prototype property of the `Cursor` object allows you to create new properties and methods of the object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.60 creates a new property and method of the `Cursor` object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to the >page.

Listing 9.60 Using the prototype *Property* to Create a New Property and Method

```
<server>

// Define the method that we prototyped.
function verifySELECT(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "SELECT"){
        return true;
    }else{
        return false;
    }
}
```

```

    }
}

// Create a new property and method of the Cursor object.
Cursor.prototype.type = null;
Cursor.prototype.isSELECT = verifySELECT;

// Create a pool of connections, a connection, and a cursor.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var currRow = myConn.cursor('SELECT * FROM employees');

// Using the prototype we defined, assign the type property.
currRow.type = "SELECT";

// Check the type of the connection to see if it is valid.
if(currRow.verifySELECT()){
    write(currRow + " has a valid type of " + currRow.type);
}else{
    write(currRow + " has an invalid type of " + currRow.type);
}

</server>

```

Cursor.unwatch()

ES3+

Syntax

cursor.unwatch(property)

Description

The `unwatch()` method of the `Cursor` object is used to turn off the watch for a particular *property*.

Example

Listing 9.61 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.61 Example of the `unwatch()` Method

```

<server>

// Define the method that we prototyped.
function verifySELECT(){

```

Listing 9.61 Continued

```
// Check to see if the type property we added is set to a valid value.
if(this.type == "SELECT"){
    return true;
}else{
    return false;
}
}

// Function that is called if property's value changes.
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Cursor object.
Cursor.prototype.type = null;
Cursor.prototype.isSELECT = verifySELECT;

// Create a pool of connections, a connection, and a cursor.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var currRow = myConn.cursor('SELECT * FROM employees');

// Using the prototype we defined, assign the type property.
currRow.type = "SELECT";

// watch property
currRow.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(currRow.verifySELECT()){
    write(currRow + " has a valid type of " + currRow.type);
}else{
    write(currRow + " has an invalid type of " + currRow.type);
}

// change value
currRow.type = "DELETE";

// turn off watch
currRow.unwatch("type");

// change value again
currRow.type = "INSERT";

</server>
```

Cursor.updateRow()

ES2+

Syntax

```
cursor.updateRow(table)
```

Description

The `updateRow()` method of the `Cursor` object uses an updateable cursor and updates the current row in the specified `table` of the cursor object. If the insert was successful, `0` is returned; otherwise, it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or `data-base` objects.

Example

Listing 9.62 creates a `Cursor` object and selects all rows. The `updateRow()` method is then used to update the current row with three specified values.

Listing 9.62 *Updating a Row from the Cursor Using the* `updateRow()` *Method*

```
<server>
```

```
// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;
client.name = request.name;
client.pwd = request.pwd;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT. Notice the
    // cursor is updateable.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT uid,name,pwd FROM employees WHERE uid = '
➤ + client.uid, true);

    // Select the row and assign values to the columns.
    currRow.next();
    currRow.uid = client.uid;
    currRow.name = client.name;
    currRow.pwd = client.pwd;
    currRow.updateRow("employees");
```

Listing 9.62 Continued

```
myConn.commitTransaction();

// Close the cursor and release the connection.
currRow.close();
myConn.release();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Cursor.watch()

ES3+

Syntax

```
cursor.watch(property, function)
```

Description

The `watch()` method of the `Cursor` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.63 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.63 Example of the `watch()` Method

```
<server>

// Define the method that we prototyped.
function verifySELECT(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "SELECT"){
        return true;
    }else{
        return false;
    }
}

// Function that is called if property's value changes.
function alertme(id, oldValue, newValue){
```

```

    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Cursor object.
Cursor.prototype.type = null;
Cursor.prototype.isSELECT = verifySELECT;

// Create a pool of connections, a connection, and a cursor.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var currRow = myConn.cursor('SELECT * FROM employees');

// Using the prototype we defined, assign the type property.
currRow.type = "SELECT";

// watch property
currRow.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(currRow.verifySELECT()){
    write(currRow + " has a valid type of " + currRow.type);
}else{
    write(currRow + " has an invalid type of " + currRow.type);
}

// change value
currRow.type = "DELETE";

</server>

```

database

ES2+

Syntax

Core object is created when the `database.connect()` method is called.

Description

The database object represents a given connection to a database. This object has only one property, the `prototype` property, which you can use to add properties to the object. Table 9.8 lists the methods associated with this object.

NOTE

If you only need a pool of connections to the database, use the `DbPool` object for your connection to initialize a pool, and then use the `Connection.connection()` method to assign a connection.

Table 9.8 *Methods of the database Object*

Method	Description
<code>beginTransaction()</code>	Begins a new SQL transaction.
<code>commitTransaction ()</code>	Commits the current SQL transaction.
<code>connect()</code>	Connects to a particular database as a particular user.
<code>connected()</code>	Tests to see whether the connection is connected to the database.
<code>cursor()</code>	Creates a cursor object for the specified SQL <code>SELECT</code> statement.
<code>disconnect()</code>	Disconnects a particular connection from the database.
<code>execute()</code>	Performs the non- <code>SELECT</code> SQL statement passed.
<code>majorErrorCode()</code>	Returns the major error code numeric value returned by the database or ODBC.
<code>majorErrorMessage()</code>	Returns the major error message string value returned by the database or ODBC.
<code>minorErrorCode()</code>	Returns the secondary error code numeric value returned by the database or ODBC.
<code>minorErrorMessage()</code>	Returns the secondary error message string value returned by the database or ODBC.
<code>rollbackTransaction()</code>	Rolls back the specified transaction.
<code>SQLTable()</code>	Formats the query results from a <code>SELECT</code> in HTML <code><table></code> format for easy writing to a client.
<code>storedProc()</code>	Creates a <code>Stproc</code> object and runs the specified stored procedure. This method was added in ES 3.0.
<code>storedProcArgs()</code>	Creates a prototype for DB2, ODBC, or Sybase stored procedures. This method was added in ES 3.0.
<code>toString()</code>	Returns a string representing the specified object.
<code>unwatch()</code>	Turns off the watch for a particular property.
<code>watch()</code>	Turns on the watch for a particular property.

Example

Listing 9.64 creates a connection to an Oracle database. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates her name. If a connection is not made, the error code and message are returned to the screen.

Listing 9.64 Creating and Using a database Object

```

<server>

// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = '
➡ + client.uid);

    // Focus on that line, change the name column for that user,
    // and update the row.
    currRow.next();
    currRow.name = client.name;
    currRow.updateRow("employees");
    myConn.commitTransaction();

    // Close the cursor and drop the connection.
    currRow.close();
    myConn.disconnect();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>

```

database.beginTransaction()

ES2+

Syntax

```
database.beginTransaction()
```

Description

The `beginTransaction()` method of the database object begins a new SQL transaction. This groups all the actions against the database together until the user exits the page or either the `commitTransaction()` or `rollbackTransaction()` methods are called. In the instance of the user exiting the page, the transaction is either committed or rolled back, depending on the setting of the commit flag when the database object instance is created.

NOTE

You cannot have nested transactions.

Example

Listing 9.65 creates a connection to an Oracle database. After the connection has been verified, the `beginTransaction()` method is called and a SQL query is performed. The results are formatted in a table with the `SQLTable()` method and written to the user's page.

Listing 9.65 Starting a New Transaction with the `beginTransaction()` Method

```
<server>

// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    // Start a new transaction and write the results to a page, formatting
    // them with the SQLTable method.
    myConn.beginTransaction();
    write(myConn.SQLTable('SELECT * FROM employees WHERE uid >= ' + client.uid));

    // Commit the transaction.
    myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

database.commitTransaction()

ES2+

Syntax

```
database.commitTransaction()
```

Description

The `commitTransaction()` method of the database object commits a new SQL transaction. This commits all the actions against the database since the last commit. If the

commit is successful, 0 is returned. If a non-zero number is returned, an error was encountered. In this case, you can use the various methods of the database object to retrieve the code and message of the error.

Example

Listing 9.66 creates a connection to an Oracle database. After the connection has been verified, the `beginTransaction()` method is called and a SQL query is performed. The results are formatted in a table with the `SQLTable()` method and written to the user's page.

Listing 9.66 *Starting a New Transaction with the* `commitTransaction()` *Method*

```
<server>

// Assign the user submitted ID to the client object as properties.
client.uid = request.uid;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    // Start a new transaction and write the results to a page, formatting
    // them with the SQLTable method.
    myConn.beginTransaction();
    write(myConn.SQLTable('SELECT * FROM employees WHERE uid >= ' + client.uid));

    // Commit the transaction.
    myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

database.connect()

ES2+

Syntax

```
database.connect(dbType, dbInstance, uid, pwd, dbName)
```

```
database.connect(dbType, dbInstance, uid, pwd, dbName, maxConn)
```

```
database.connect(dbType, dbInstance, uid, pwd, dbName, maxConn, commitFlag)
```

Description

The connect method of the database object is the actual method that connects to a database given the parameters passed. Before you open a connection to a database and have the ability to run queries against it, you should create an instance of this object. Each parameter is defined in Table 9.9.

Table 9.9 *Parameters of the connect Method*

Parameter	Description
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC it is the DSN entry name.
uid	The username or ID you want the connection to connect as.
pwd	The password for the user you are connecting as.
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections this should be a blank, "", string. In Oracle, the name of the database for these connections is set up in the tnsnames.ora file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the dbInstance.
maxConn	The maximum number of connections to the pool. This is effectively the number of connections the pool will open to the database.
commitFlag	This flag determines whether a pending transaction is committed when connection is released. If it is set to false, the transaction is rolled back. If it is set to true, it is committed.

Depending on your database, it is possible to create an instance of this object by passing a limited set of these parameters. See your database documentation for this information.

Example

Listing 9.67 creates a connection to an Oracle database. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates her name. If a connection is not made, the error code and message are returned to the screen.

Listing 9.67 *Connecting to a Database Using the connect() Method*

```
<server>

// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;

// Open a connection.
```

```

var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS", true);

if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = '
➤ + client.uid);

    // Focus on that line, change the name column for that user,
    // and update the row.
    currRow.next();
    currRow.name = client.name;
    currRow.updateRow("employees");
    myConn.commitTransaction();

    // Close the cursor and the connection.
    currRow.close();
    myConn.disconnect();

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>

```

database.connected()

ES2+

Syntax

```
database.connected()
```

Description

The `connected()` method of the database object tells whether the connection to the database is still active.

Example

Listing 9.68 creates a connection to a database. If the connection is made, any code within that section is executed. If the connection fails, the error is written to the page.

Listing 9.68 *Testing a Connection with the* `connected()` *Method*

```

<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

```

Listing 9.69 Continued

```
if (myConn.connected()) {  
  
    // You are connected, so perform any tasks here.  
  
}else{  
  
    // There was an error connecting to the database.  
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());  
}  
  
</server>
```

database.cursor()

ES2+

Syntax

```
database.cursor(sql)  
database.cursor(sql, boolean)
```

Description

The `cursor()` method of the `database` object creates a `Cursor` object that can be used to run SQL queries against the database. The method takes the *sql* statement as a parameter, as well as an optional *boolean* value that specifies whether the cursor is updateable.

Example

Listing 9.69 shows how you would run a query against the database using the `cursor()` method. The `while` loop is used to write the results to the user's page.

Listing 9.69 Using the `cursor()` Method to Run a Query Against the Database

```
<server>  
  
// Set the query to run.  
var mySQL = myConn.cursor('SELECT name,title FROM employees');  
  
// Iterate through the results and write them to the page.  
while(mySQL.next()){  
    write(mySQL.name + ': ' + mySQL.title + '<br>');  
}  
  
</server>
```

database.disconnect()

ES2+

Syntax

```
database.disconnect()
```

Description

The disconnect method of the database object disconnects a connection to a database.

Example

Listing 9.70 creates a connection to an Oracle database. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates her name. If a connection is not made, the error code and message are returned to the screen. After the processing has been completed, the connection is dropped using the disconnect() method.

Listing 9.70 *Disconnecting from a Database Using the* disconnect() *Method*

```
<server>
```

```
// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS", true);

if(myConn.connected()){

    // Start a new SQL transaction to perform a SELECT.
    myConn.beginTransaction();
    var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = '
    + client.uid);

    // Focus on that line, change the name column for that user,
    // and update the row.
    currRow.next();
    currRow.name = client.name;
    currRow.updateRow("employees");
    myConn.commitTransaction();

    // Close the cursor and the connection.
    currRow.close();
    myConn.disconnect();
```

Listing 9.70 Continued

```
// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

database.execute()ES2+

Syntax

```
database.execute(statement)
```

Description

The `execute()` method of the database object enables your application to execute a *DDL (Data Definition Language)* or *DML (Data Manipulation Language)* query, which does not return a Cursor, supported by your database. This includes statements such as `CREATE`, `ALTER`, and `DROP`.

 NOTE

Be sure to use SQL that conforms to your database.

Example

Listing 9.71 deletes all rows with a UID less than the number passed to the script.

Listing 9.71 Using the `execute()` Method to Run DML Queries

```
<server>

// Assign the UID passed to the client object.
client.uid = request.uid;

// Execute a DELETE based on the UID passed.
myConn.execute('DELETE FROM employees WHERE uid < ' + client.uid);

</server>
```

database.majorErrorCode()ES2+

Syntax

```
database.majorErrorCode()
```


Description

The `majorErrorCode()` method of the database object contains the ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.72 shows how you would create a connection and test for a successful connection. If the test fails, the `majorErrorCode()` is used when writing the error to the page.

Listing 9.72 Using `majorErrorCode()` to Retrieve a Database Connection Error

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

database.majorErrorMessage()

ES2+

Syntax

```
database.majorErrorMessage()
```

Description

The `majorErrorMessage()` method of the database object contains the ODBC or database string error message that is returned if an error occurs.

Example

Listing 9.73 shows how you would create a connection and test for a successful connection. If the test fails, the `majorErrorMessage()` is used when writing the error to the page.

Listing 9.73 Using `majorErrorMessage()` to Retrieve a Database Connection Error

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

database.minorErrorCode()ES2+

Syntax

```
database.minorErrorCode()
```

Description

The `minorErrorCode()` method of the database object contains the secondary ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.74 shows how you would create a connection and test for a successful connection. If the test fails, the `minorErrorCode()` is used when writing the secondary error to the page.

Listing 9.74 Using `minorErrorCode()` to Retrieve a Secondary Database Connection Error

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myConn.connected()) {
```

```
// You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.minorErrorCode()+'): '+myConn.minorErrorMessage);
}

</server>
```

database.minorErrorMessage()

ES2+

Syntax

```
database.minorErrorMessage()
```

Description

The `minorErrorMessage()` method of the database object contains the secondary ODBC or database string error message that is returned if an error occurs.

Example

Listing 9.75 shows how you would create a connection and test for a successful connection. If the test fails, the `minorErrorMessage()` is used when writing the secondary error to the page.

Listing 9.75 Using `minorErrorMessage()` to Retrieve a Secondary Database Connection Error

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.minorErrorCode()+'): '+myConn.minorErrorMessage);
}

</server>
```

database.prototype

ES2+

Syntax

```
database.prototype.method = name  
database.prototype.property = value
```

Description

The prototype property of the database object allows you to add methods and properties to the database object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.76 creates a new property and method of the database object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect (which it is), an error message is written to the page.

Listing 9.76 *Using the prototype Property to Create a New Property and Method*

```
<server>
```

```
// Define the method that we prototyped.  
function verifyODBC(){  
  
    // Check to see if the type property we added is set to a valid value.  
    if(this.type == "ODBC"){  
        return true;  
    }else{  
        return false;  
    }  
}  
  
// Create a new property and method of the database object.  
database.prototype.type = null;  
database.prototype.isODBC = verifyODBC;  
  
// Open a connection.  
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");  
  
// Using the prototype we defined, assign the type property.  
myConn.type = "Oracle";  
  
// Check the type of the connection to see if it is valid.  
if(myConn.isODBC()){  
    write(myConn + " has a valid type of " + myConn.type);  
}else{
```

```

    write(myConn + " has an invalid type of " + myConn.type);
}

</server>

```

database.rollbackTransaction()

ES2+

Syntax

```
database.rollbackTransaction()
```

Description

The `rollbackTransaction()` method of the `database` object will undo all actions performed since the last `beginTransaction()` method call.

NOTE

You cannot have nested transactions.

Example

Listing 9.77 takes a *commit* field sent to the application from the user. If this evaluates to true, the transaction is committed. If not, it is rolled back.

Listing 9.77 *Rolling Back a Transaction with the* `rollbackTransaction()` *Method*

```

<server>

// See if the user wants to commit the last transaction.
client.commit = request.commit;

if(client.commit = "YES"){

    // Commit the transaction.
    myConn.commitTransaction();

}else{

    // Rollback the transaction.
    myConn.rollbackTransaction();
}

</server>

```

database.SQLite()

ES2+

Syntax

```
database.SQLite(sql)
```

Description

The SQLite() method of the database object takes a *sql* SELECT statement as a parameter and executes this query through the connection from which it was called. It returns the results formatted in an HTML table for easy writing to a client's page. This is a simple table in the following format:

```
<table border="1">
<tr>
<th>column 1</th>
<th>column 2</th>
...
<th>column N</th>
</tr>
<tr>
<td>value 1 of column 1</td>
<td>value 1 of column 2</td>
...
<td>value 1 of column N</td>
</tr>
<tr>
<td>value 2 of column 1</td>
<td>value 2 of column 2</td>
...
<td>value 2 of column N</td>
</tr>
...
</table>
```

Example

Listing 9.78 runs a user passed query and formats the result using the SQLite() method. This information is then written to the user's page.

Listing 9.78 Using the SQLite() Method to Format the Result of a SELECT Query

```
<server>
```

```
// Assign the user submitted query to the client object.
client.sql = request.sql;
```

```
// Open a connection.
```

```

var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    // Start a new transaction and write the results to a page, formatting
    // them with the SQLTable method.
    myConn.beginTransaction();
    write(myConn.SQLTable(client.sql));

    // Commit the transaction.
    myConn.commitTransaction();

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>

```

database.storedProc()

ES3+

Syntax

```

database.storedProc(procName)
database.storedProc(procName, arg1, arg2, ... , argN)

```

Description

The `storedProc()` method of the `database` object creates a `Stproc` object that allows you to execute a database-specific stored procedure using the connection from which it was invoked.

As shown in the syntactical definition, you can also pass any arguments needed to the method for processing. If you are using a stored procedure that requires arguments, or if you want to have the procedure run using default arguments, you must pass `/Default/` as the argument. The following shows an example of passing a default value:

```
var myStproc = myConn.storedProc("sp_employees", "/Default/");
```

The scope of this procedure is restricted to the current page. Any methods of the `Stproc` object must be invoked on the current page. If this is not possible, a new object will have to be created on subsequent pages to access the properties needed.

Example

Listing 9.79 creates a connection to a database. When the connection has been verified, the `storedProc()` method is used to invoke the fictitious `sp_employees` stored procedure.

Listing 9.79 *Using the `storedProc()` Method to Invoke a Stored Procedure on a Database*

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    myConn.beginTransaction();

    // Run the stored procedure.
    var myStproc = myConn.storedProc("sp_employees");

    // Commit the transaction.
    myConn.commitTransaction();

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server> >
```

database.storedProcArgs()

ES3+

Syntax

```
database.storedProcArgs(procName)
database.storedProcArgs(procName, type1, type2, ... , typeN)
```

Description

The `storedProcArgs()` method of the `database` object creates a `Stproc` object that allows you to execute a database-specific stored procedure using the connection from which it was invoked on DB2, ODBC, and Sybase databases. If this method is invoked on Informix or Oracle databases, it has no effect. The difference between this method and the `storedProc()` method is that this method takes a type as a parameter for the

arguments passed. These types can be IN, OUT, or INOUT. The following shows an example of passing these types:

```
var myStproc = myConn.storedProc("sp_employees", "INOUT", "OUT");
```

The scope of this procedure is restricted to the current page. Any methods of the Stproc object must be invoked on the current page. If this is not possible, a new object will have to be created on subsequent pages to access the properties needed.

Example

Listing 9.80 creates a connection to a database. After the connection has been verified, the storedProc() method is used to invoke the fictitious *sp_employees* stored procedure, and the storedProcArgs() method is used to specify the argument types.

Listing 9.80 Using the storedProcArgs() Method to Set the Argument Types of a Stored Procedure

```
<server>
```

```
// Open a connection.
```

```
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
if(myConn.connected()){
```

```
    myConn.beginTransaction();
```

```
    // Run the stored procedure.
```

```
    var myStprocArgs = myConn.storedProcArgs("sp_employees", "IN", "INOUT");
```

```
    var myStproc = myConn.storedProc("sp_employees", 3, "%John%");
```

```
    // Commit the transaction.
```

```
    myConn.commitTransaction();
```

```
// If the connection fails, write an error message.
```

```
}else{
```

```
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
```

```
}
```

```
// Release the connection.
```

```
myConn.release();
```

```
</server>
```

database.toString()

ES2+

Syntax

```
database.toString()
```

Description

The `toString()` method of the database object returns a text value of the object. When invoked on an instance of a database object, the string is returned in the following format:

```
"dbName" "uid" "dbType" "dbInstance"
```

If the parameter is unknown, an empty string is returned. Table 9.10 contains the value of these returned values.

Table 9.10 *Return Values of the `toString()` Method*

Method	Description
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections, this is a blank, "", string. In Oracle, the name of the database for these connections are set up in the <code>tnsnames.ora</code> file and are defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the <code>dbInstance</code> .
uid	The username or ID you want the connections to connect as.
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC, it is the DSN entry name.

Example

Listing 9.81 creates an instance of the database object. Once created, the `write()` method is used to write its string value to the page.

Listing 9.81 *Write the Results of Calling the `toString()` Method to a Page*

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    // Write the string value of the object to the page.
    write(myConn.toString());

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}
```

```
// Release the connection.
myConn.release();
```

```
</server>
```

database.unwatch()

ES3+

Syntax

```
database.unwatch(property)
```

Description

The `unwatch()` method of the database object is used to turn off the watch for a particular *property*.

Example

Listing 9.82 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.82 Example of the `unwatch()` Method

```
<server>
```

```
// Define the method that we prototyped.
function verifyODBC(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "ODBC"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + " ) changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the database object.
database.prototype.type = null;
database.prototype.isODBC = verifyODBC;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

Listing 9.82 Continued

```
// Using the prototype we defined, assign the type property.
myConn.type = "Oracle";

// Check the type of the connection to see if it is valid.
if(myConn.isODBC()){
    write(myConn + " has a valid type of " + myConn.type);
}else{
    write(myConn + " has an invalid type of " + myConn.type);
}

// watch property
myConn.watch("type", alertme);

// change value
myConn.type = null;

// turn off watch
myConn.unwatch("type");

// change value again
myConn.type = "Sybase";

</server>
```

database.watch()

ES3+

Syntax

```
database.watch(property, function)
```

Description

The `watch()` method of the database object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.83 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.83 Example of the watch() Method

```
<server>

// Define the method that we prototyped.
function verifyODBC(){
```

```
// Check to see if the type property we added is set to a valid value.
if(this.type == "ODBC"){
    return true;
}else{
    return false;
}
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the database object.
database.prototype.type = null;
database.prototype.isODBC = verifyODBC;

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Using the prototype we defined, assign the type property.
myConn.type = "Oracle";

// Check the type of the connection to see if it is valid.
if(myConn.isODBC()){
    write(myConn + " has a valid type of " + myConn.type);
}else{
    write(myConn + " has an invalid type of " + myConn.type);
}

// watch property
myConn.watch("type", alertme);

// change value
myConn.type = null;

</server>
```

DbPool

ES3+

Syntax

```
new DbPool()
```

```
new DbPool(dbType, dbInstance, uid, pwd, dbName)
```

```
new DbPool(dbType, dbInstance, uid, pwd, dbName, maxConn)

new DbPool(dbType, dbInstance, uid, pwd, dbName, maxConn, commitFlag)
```

Description

The DbPool object is an object that holds a “pool” of connections to a database. Before you open a connection to a database and have the ability to run queries against it, you should create an instance of this object. After the instance is created, connections can be obtained from the pool as needed. The pool object itself takes all the parameters necessary to make the connection. It is possible to create a pool without specifying any parameters; however, you must pass the parameters when the first connection is attempted.

The creation of a DbPool object is done using the format defined in the syntax definition. Each parameter is defined in Table 9.11.

Table 9.11 Parameters of the DbPool Object

Parameter	Description
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC, it is the DSN entry name.
uid	The username or ID you want the connections to connect as.
pwd	The password for the user you are connecting as.
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections this should be a blank, "", string. In Oracle, the name of the database for these connections is set up in the tnsnames.ora file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the dbInstance.
maxConn	The maximum number of connections to the pool. This is effectively the number of connections the pool will open to the database.
commitFlag	This flag determines whether a pending transaction is committed when connection is released. If it is set to false, the transaction is rolled back. If it is set to true, it is committed.

Depending on your database, it is possible to create an instance of this object by passing a limited set of these parameters, as well as passing none. The object itself has the methods listed in Table 9.12.

Table 9.12 *Methods of the DbPool Object*

Method	Description
<code>connect()</code>	Connects to a particular pool of database connections
<code>connected()</code>	Tests to see whether the pool is still connected to the database
<code>connection()</code>	Obtains an available connection from the pool
<code>DbPool()</code>	Creates the pool of connections to a database
<code>disconnect()</code>	Disconnects all connections in the pool from the database
<code>majorErrorCode()</code>	Returns the major error code numeric value returned by the database or ODBC
<code>majorErrorMessage()</code>	Returns the major error message string value returned by the database or ODBC
<code>minorErrorCode()</code>	Returns the secondary error code numeric value returned by the database or ODBC
<code>minorErrorMessage()</code>	Returns the secondary error message string value returned by the database or ODBC
<code>storedProcArgs()</code>	Creates a prototype for DB2, ODBC, or Sybase stored procedures
<code>toString()</code>	Returns a string representing the specified object
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property

Example

Listing 9.84 creates a pool of connections to an Oracle database and initializes a connection from that pool. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates her name. If a connection is not made, the error code and message are returned to the screen.

Listing 9.84 *Creating and Using a DbPool Object*

```
<server>
```

```
// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;
```

```
// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn){
```

Listing 9.84 Continued

```
// Start a new SQL transaction to perform a SELECT.
myConn.beginTransaction();
var currRow = myConn.cursor('SELECT * FROM employees WHERE uid = '
➤ + client.uid);

// Focus on that line, change the name column for that user,
// and update the row.
currRow.next();
currRow.name = client.name;
currRow.updateRow("employees");

// Close the cursor.
currRow.close();

// If the connection fails, write an error message.
}else{
  write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

DbPool.connect()

ES3+

Syntax

```
dbpool.connect(dbType, dbInstance, uid, pwd, dbName)

dbpool.connect(dbType, dbInstance, uid, pwd, dbName, maxConn)

dbpool.connect(dbType, dbInstance, uid, pwd, dbName, maxConn, commitFlag)
```

Description

The connect method of the DbPool object is used to connect to a database when the connection was not made with the initialization of the original DbPool object. The method takes all the parameters necessary to connect to the database. Each parameter is defined in Table 9.9.

Table 9.13 Parameters of the connect() Method

Parameter	Description
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC it is the DSN entry name.
uid	The username or ID you want the connections to connect as.
pwd	The password for the user you are connecting as.

Parameter	Description
dbName	The name of the database you want to log into. For Oracle, DB2, and ODBC connections this should be a blank, "", string. In Oracle, the name of the database for these connections is set up in the <code>tnsnames.ora</code> file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the <code>dbInstance</code> .
maxConn	The maximum number of connections to the pool. This is effectively the number of connections the pool will open to the database.
commitFlag	This flag determines whether a pending transaction is committed when connection is released. If it is set to <code>false</code> , the transaction is rolled back. If it is set to <code>true</code> , it is committed.

Depending on your database, it is possible to create an instance of this object by passing a limited set of these parameters. See your database documentation for this information.

Example

Listing 9.85 creates a connection pool. The `connect()` method is then called to open the pool to an Oracle database. If a connection is not made, the error code and message are returned to the screen.

Listing 9.85 Connecting to a Database Using the `connect()` Method

```
<server>
```

```
// Assign the user submitted ID and name to the client object as properties.
client.uid = request.uid;
client.name = request.name;

// Create a pool of connections.
var myPool = new DbPool();

// Create a connection for the pool.
myPool.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if(myConn.connected()){

    // Do any database stuff here.
```

Listing 9.85 Continued

```
// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

DbPool.connected()

ES3+

Syntax

```
dbpool.connected()
```

Description

The `connected()` method of the `DbPool` object tells whether the pool of connections to the database is still connected.

Example

Listing 9.86 creates a pool of connections and pulls a connection from the pool for processing. If the connection is made, any code within that section is executed. If the connection fails, the error is written to the page.

Listing 9.86 Testing a Connection with the `connected()` Method

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

DbPool.connection()

ES3+

Syntax

```
dbpool.connection(name, seconds);
```

Description

The `connection()` method of the `DbPool` object pulls a connection from the pool. The connection is returned from the method and can be stored in a variable to be used for processing.

The method takes two parameters. The first parameter is a *name*, which is a name you can give your connection. Because you actually store the connection in a variable, this name's primary function becomes one for debugging purposes. The second parameter is a *seconds* value for the number of seconds you give the instance to connect.

Example

Listing 9.87 creates a pool of connections to an Oracle database and initializes a connection from that pool. It takes a user's UID and name that was passed in, runs a query (based on the UID) against the database to find that user's information, and updates her name. If a connection is not made, the error code and message are returned to the screen.

Listing 9.87 *Creating and Using a connection() Method*

```
<server>
```

```
// Create a pool of connections.
```

```
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
// Open a connection from the pool. Give error if connection could
```

```
// not be made.
```

```
var myConn = myPool.connection('Employees', 15);
```

```
if (myConn.connected()) {
```

```
    // You are connected, so perform any tasks here.
```

```
}else{
```

```
    // There was an error connecting to the database.
```

```
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
```

```
}
```

```
</server>
```

DbPool.DbPool()

ES3+

Syntax

```
new DbPool()  
  
new DbPool(dbType, dbInstance, uid, pwd, dbName)  
  
new DbPool(dbType, dbInstance, uid, pwd, dbName, maxConn)  
  
new DbPool(dbType, dbInstance, uid, pwd, dbName, maxConn, commitFlag)
```

Description

The DbPool() method of the DbPool object is the underlying method that creates a “pool” of connections to a database. The creation of a DbPool object is done using the format defined in the syntax definition. Each parameter is defined in Table 9.14.

Table 9.14 Parameters of the DbPool() Method

Parameter	Description
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC it is the DSN entry name.
uid	The username or ID you want the connections to connect as.
pwd	The password for the user you are connecting as.
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections this should be a blank, "", string. In Oracle, the name of the database for these connections is set up in the tnsnames.ora file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the dbInstance.
maxConn	The maximum number of connections to the pool. This is effectively the number of connections the pool will open to the database.
commitFlag	This flag determines whether a pending transaction is committed when connection is released. If it is set to false, the transaction is rolled back. If it is set to true, it is committed.

Depending on your database, it is possible to create an instance of this object by passing a limited set of these parameters.

Example

Listing 9.88 creates a pool of connections and pulls a connection from the pool for processing. If the connection is made, any code within that section is executed. If the connection fails, the error is written to the page.

Listing 9.88 *The DbPool() Method is the Underlying Method Used When a DbPool Object Instance Is Created*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if (myConn.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

DbPool.disconnect()

ES3+

Syntax

```
dbpool.disconnect()
```

Description

The `disconnect()` method of the `DbPool` object disconnects all connections to a database within that pool.

Example

Listing 9.89 creates a connection to an Oracle database. The next line drops the connection by using the `disconnect()` method.

Listing 9.89 *Disconnecting from a Database Using the disconnect() Method*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

Listing 9.89 Continued

```
// Drop the connections.
myPool.disconnect();

</server>
```

DbPool.majorErrorCode()

ES3+

Syntax

```
dbpool.majorErrorCode()
```

Description

The `majorErrorCode()` method of the `DbPool` object contains the ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.90 shows how you would create a pool of connections and test for the connection. If the test fails, the `majorErrorCode()` is used when writing the error to the page.

Listing 9.90 Using `majorErrorCode()` to Retrieve a Database Connection Error

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myPool.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myPool.majorErrorCode()+'): '+myPool.majorErrorMessage);
}

</server>
```

DbPool.majorErrorMessage()

ES3+

Syntax

```
dbpool.majorErrorMessage()
```

Description

The `majorErrorMessage()` method of the `DbPool` object contains the ODBC or database string error message that is returned if an error occurs.

Example

Listing 9.91 shows how you would create a pool of connections and test for the connection. If the test fails, the `majorErrorMessage()` method is used to write the error to the page.

Listing 9.91 *Using `majorErrorMessage()` to Retrieve a Database Connection Error*

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myPool.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myPool.majorErrorCode()+'): '+myPool.majorErrorMessage);
}

</server>
```

DbPool.minorErrorCode()

ES3+

Syntax

```
dbpool.minorErrorCode()
```

Description

The `minorErrorCode()` method of the `DbPool` object contains the secondary ODBC or database numeric error code that is returned if an error occurs.

Example

Listing 9.92 shows how to create a pool of connections and test for the connection. If the test fails, the `minorErrorCode()` method is used to write the secondary error to the page.

Listing 9.92 Using `minorErrorCode()` to Retrieve a Secondary Database Connection Error

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myPool.connected()) {

    // You are connected, so perform any tasks here.

}else{

    // There was an error connecting to the database.
    write('Error ('+myPool.minorErrorCode()+'): '+myPool.minorErrorMessage);
}

</server>
```

DbPool.minorErrorMessage()ES3+

Syntax

```
dbpool.minorErrorMessage()
```

Description

The `minorErrorMessage()` method of the `DbPool` object contains the secondary ODBC or database string error message that is returned if an error occurs.

Example

Listing 9.93 shows how to create a pool of connections and test for the connection. If the test fails, the `minorErrorMessage()` method is used to write the secondary error to the page.

Listing 9.93 Using `minorErrorMessage()` to Retrieve a Secondary Database Connection Error

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if (myPool.connected()) {

    // You are connected, so perform any tasks here.

}else{
```



```
// There was an error connecting to the database.
write('Error ('+myPool.minorErrorCode()+'): '+myPool.minorErrorMessage);
}

</server>
```

DbPool.prototype

ES3+

Syntax

```
dbpool.prototype.method = name
dbpool.prototype.property = value
```

Description

The prototype property of the DbPool object allows you to add methods and properties to the DbPool object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.94 creates a new property and method of the DbPool object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to the page.

Listing 9.94 Using the prototype Property to Create a New Property and Method

```
<server>

// Define the method that we prototyped.
function verifyOracle(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "Oracle"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the DbPool object.
DbPool.prototype.type = null;
DbPool.prototype.isOracle = verifyOracle;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

Listing 9.94 Continued

```
// Using the prototype we defined, assign the type property.
myPool.type = "Oracle";

// Check the type of the connection to see if it is valid.
if(myPool.isOracle()){
    write(myPool + " has a valid type of " + myPool.type);
}else{
    write(myPool + " has an invalid type of " + myPool.type);
}

</server>
```

DbPool.storedProcArgs()

ES3+

Syntax

```
database.storedProcArgs(procName)
database.storedProcArgs(procName, type1, type2, ... , typeN)
```

Description

The `storedProcArgs()` method of the database object creates a `Stproc` object that allows you to execute a database-specific stored procedure using the connection from which it was invoked on DB2, ODBC, and Sybase databases. If this method is invoked on Informix or Oracle databases, it has no affect. The difference between this method and the `storedProc()` method is that this method takes a type as a parameter for the arguments passed. These types can be `IN`, `OUT`, or `INOUT`. The following shows an example of passing these types:

```
var myStproc = myConn.storedProc("sp_employees", "INOUT", "OUT");
```

The scope of this procedure is restricted to the current page. Any methods of the `Stproc` object must be invoked on the current page. If this is not possible, a new object will have to be created on subsequent pages to access the properties needed.

Example

Listing 9.95 creates a pool of connections to a database. The `storedProc()` method is used to invoke the fictitious `sp_employees` stored procedure, and the `storedProcArgs()` method is used to specify the argument types.

Listing 9.95 Using the `storedProcArgs()` Method to Set the Argument Types of a Stored Procedure

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
// Set the stored procedure arguments.
var myStprocArgs = myPool.storedProcArgs("sp_employees", "IN", "INOUT");
var myStproc = myPool.storedProc("sp_employees", 3, "%John%");

</server>
```

DbPool.toString()

ES3+

Syntax

```
dbpool.toString()
```

Description

The `toString()` method of the `DbPool` object returns a text value of the object. When invoked on an instance of a `DbPool` object, the string is returned in the following format:

```
"dbName" "uid" "dbType" "dbInstance"
```

If the parameter is unknown, an empty string is returned. Table 9.15 contains the value of these returned values.

Table 9.15 *Return Values of the `toString()` Method*

Method	Description
dbName	The name of the database you want to log in to. For Oracle, DB2, and ODBC connections this is a blank, "", string. In Oracle, the name of the database for these connections is set up in the <code>tnsnames.ora</code> file and is defined by the DSN for ODBC connections. DB2 does not have a database name and is referenced only by the <code>dbInstance</code> .
uid	The username or ID you want the connections to connect as.
dbType	The type of database it is. Possible values are ORACLE, SYBASE, INFORMIX, DB2, or ODBC.
dbInstance	This is the instance name of the database. For ODBC, it is the DSN entry name.

Example

Listing 9.96 creates an instance of the `DbPool` object. Once created, the `write()` method is used to write its string value to the page.

Listing 9.96 Write the Results of Calling the toString() Method to a Page

```
<server>

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);
if(myConn.connected()){

    // Write the string value of the object to the page.
    write(myPool.toString());

    // If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection.
myConn.release();

</server>
```

DbPool.unwatch()

ES3+

Syntax

```
dbpool.unwatch(property)
```

Description

The `unwatch()` method of the `DbPool` object is used to turn off the watch for a particular *property*.

Example

Listing 9.97 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.97 Example of the unwatch() Method

```
<server>

// Define the method that we prototyped.
function verifyOracle(){
```

```
// Check to see if the type property we added is set to a valid value.
if(this.type == "Oracle"){
    return true;
}else{
    return false;
}
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the DbPool object.
DbPool.prototype.type = null;
DbPool.prototype.isOracle = verifyOracle;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Using the prototype we defined, assign the type property.
myPool.type = "Oracle";

// watch property
myPool.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myPool.isOracle()){
    write(myPool + " has a valid type of " + myPool.type);
}else{
    write(myPool + " has an invalid type of " + myPool.type);
}

// change value
myPool.type = null;

// turn off watch
myPool.unwatch("type");

// change value again
myPool.type = "Sybase";

</server>
```

DBPool.watch()

ES3+

Syntax

```
dbpool.watch(property, function)
```

Description

The `watch()` method of the `DBPool` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.98 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.98 Example of the `watch()` Method

```
<server>
```

```
// Define the method that we prototyped.
function verifyOracle(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "Oracle"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the DbPool object.
DbPool.prototype.type = null;
DbPool.prototype.isOracle = verifyOracle;

// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Using the prototype we defined, assign the type property.
myPool.type = "Oracle";
```

```
// watch property
myPool.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myPool.isOracle()){
    write(myPool + " has a valid type of " + myPool.type);
}else{
    write(myPool + " has an invalid type of " + myPool.type);
}

// change value
myPool.type = null;

</server>
```

debug()

ES2+

Syntax

```
debug(expression)
debug(variable)
```

Description

The debug function is a top-level function that is not associated with any core object. This function is used to display the value of an expression or variable in the Trace Information window when running the application in the JavaScript Application Manager's debug window.

Example

Listing 9.99, when run in the JavaScript Application Manager's debugger, will display the value of the *request.name* when encountered.

Listing 9.99 Using the debug() Function to Write Information to the Trace Information Window

```
<server>

// Display the value of the name passed in the request
// to the application.
debug(request.name);

</server>
```

deleteResponseHeader()

ES3+

Syntax

```
deleteResponseHeader(key)
```

Description

The `deleteResponseHeader()` function is a top-level function and is not associated with any core object. This function is used to delete fields in the HTTP header before it is sent back to the client. Because of when the actual header is sent in relation to the body of the data, you should be sure to delete these fields before you call the `flush()` or `redirect()` functions.

NOTE

The JavaScript runtime engine flushes the output buffer after 64KB of content has been generated. You should be sure to call the `deleteResponseHeader()` function before this time.

Example

Listing 9.100 shows how you can delete the content-type header field before it is sent back to the browser.

Listing 9.100 Using the `deleteResponseHeader()` Function to Delete the content.type of a File Being Sent to a Browser

```
<server>

// Delete a field to the header.
deleteResponseHeader("content-type");

</server>
```

File()

ES2+

Syntax

```
new File(path)
```

Description

The `File` object allows you to perform various tasks such as reading and writing to a file on your disk. The object itself has many methods to use and a `prototype` property that allows a programmer to create new properties and methods of the object. Table 9.16 lists the methods accessible and a brief description of each. An instance of this object is created by simply passing the *path* of the file you want to create or read.

Table 9.16 *Methods of the File Object*

Method	Description
<code>byteToString()</code>	Converts the byte number passed into its string equivalent
<code>clearError()</code>	Clears the <code>File.eof()</code> and <code>File.error()</code> error status
<code>close()</code>	Closes the file you opened
<code>eof()</code>	Returns <code>true</code> if you are at the end of the file you have opened
<code>error()</code>	Returns the current error
<code>exists()</code>	Checks to see whether the file you want to process exists
<code>flush()</code>	Writes the contents of the current buffer to the file
<code>getLength()</code>	Returns the length of the file
<code>getPosition()</code>	Returns your current position within a file
<code>open()</code>	Opens the file
<code>read()</code>	Reads the specified number of characters into a string
<code>readByte()</code>	Reads the next byte, or character, in the file
<code>readLn()</code>	Reads the current line, starting at your current position, into a string
<code>setPosition()</code>	Sets your position in a file
<code>stringToByte()</code>	Converts the string passed into its byte number equivalent
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property
<code>write()</code>	Writes a string to the file you opened
<code>writeByte()</code>	Writes a byte of data to a binary file you opened
<code>writeln()</code>	Writes a string and a carriage return to the file you opened

The usage of the `File` object is very straightforward. The methods provided allow you to perform the various tasks needed on the files on your file system. Part of this functionality of working with these files is to allow programmers to specify how they want to open the files. A file can be opened to read, write, append, or open in binary mode. These options are specified in the `open()` method in the following form:

```
myFile.open("option");
```

Table 9.17 gives a list and description of these options.

Table 9.17 *Options of the open() Method*

Option	Description
<code>a</code>	This option opens a file for appending. If the file does not exist, it is created. This method always returns <code>true</code> .
<code>a+</code>	This option opens a file for reading and appending. If the file does not exist, it is created. This method always returns <code>true</code> .
<code>r</code>	This option opens a file for reading. If the file exists, the method returns <code>true</code> ; otherwise, it returns <code>false</code> .
<code>r+</code>	This option opens a file for reading and writing. If the file exists, the method returns <code>true</code> ; otherwise, it returns <code>false</code> . Reading and writing start at the beginning of the file.

Table 9.17 Continued

Option	Description
w	This option opens a file for writing. If the file does not exist, it is created. If it does exist, it is overwritten. This method always returns true.
w+	This option opens a file for reading and writing. If the file does not exist, it is created. If it does exist, it is overwritten. This method always returns true.
optionb	Appending b to the end of any of these options specifies that you want to perform the operation in binary mode.

Example

Listing 9.101 displays an option menu that allows a user to select a file to read. When the form is submitted, the script reads the file and displays its contents on a page.

Listing 9.101 Using the File Object

```
<html>
<head>
  <title> Using the File object</title>
</head>
<body>
<server>

// See if they have submitted or just need the form1
if(request.method == "POST"){

  // Create an instance of the File object and pass it the file
  // the user specified they wanted to view.
  var myLog = new File(request.file);

  // Try to open the file.
  if(!myLog.open("r")){

    // If there was an error, tell the user.
    write("There was an error opening the file: " + request.file);
  }else{

    // If there was not an error, then open the file and display it.
    write('<h3>The contents of ' + request.file + ' are as follows:</h3>');
    while(!myLog.eof()){
      write(myLog.readln());
    }
  }
}
}else{
```

```
// If this page was called then write the select box to the page for
// the user to use to select which log they want to see.

write('<form name="myForm" method="post">');
write('<select name="file">');
write('<option value="/logs/admin.log">Admin Log</option>');
write('<option value="/logs/user.log">User Log</option>');
write('<option value="/logs/error.log">Error Log</option>');
write('</select>');
write('<input type="submit" value="View Log">');
write('</form>');
}

</server>
</body>
</html>
```

File.toString()

ES2+

Syntax

File.toString(num)

Description

The `toString()` method of the `File` object is used to convert the numeric value passed to its ASCII equivalent. If the method is not passed a number, an empty string is returned.

Example

Listing 9.102 opens two files, one for reading and the other for appending. Bytes are then read using the `readByte()` method from the first file, converted back to string characters using the `toString()` method, and written to the second file. Both files are closed when the process has completed.

Listing 9.102 *Using the `toString()` Method to Convert the Bytes Read into Strings*

```
<server>

// Open a log file and a summary file.
var myLog = new File("/data/logs/today.log");
var mySummary = new File("/data/logs/summary.log");

// Open the log file for reading and the summary file for
// appending.
myLog.open("r");
```

Listing 9.102 Continued

```
mySummary.open("a");

// Append the contents of the log file to the summary file.
while (!myLog.eof()){
    myBytes = File.toString(myLog.readByte());
    mySummary.write(myBytes);
}

// Close the files.
myLog.close();
mySummary.close();

</server>
```

File.clearError()

ES2+

Syntax

```
file.clearError()
```

Description

The `clearError()` method of the `File` object clears the file error status and the value returned by the `eof()` method.

Example

Listing 9.103 opens a file for reading. If the operation returned an error, the error is written to the page. If there was an error, it is cleared after writing it.

Listing 9.103 Using the `clearError()` Method to Clear File Errors

```
<server>

// Open a log file.
var myLog = new File("/data/logs/today.log");

// Open the log file for reading.
myLog.open("r");

if (myLog.error() != 0) {

    // Perform actions on file.

}else{
```

```
// Write out the error.
write('Error: ' + myLog.error());

// Clear the error.
myLog.clearError()
}

// Close the file.
myLog.close();

</server>
```

File.close()

ES2+

Syntax

```
file.close()
```

Description

The `close()` method of the `File` object closes the file on which it has been invoked. This method returns `true` if it was successful and `false` if it was unsuccessful.

Example

Listing 9.104 shows how to open a file and then close it.

Listing 9.104 Closing a File with the `close()` Method

```
<server>

// Open a log file.
var myLog = new File("/data/logs/today.log");

// Open the log file for reading.
myLog.open("r");

// Close the file.
myLog.close();

</server>
```

File.constructor

ES2+

Syntax

```
file.constructor
```

Description

The constructor property of the `File` object specifies the function that creates the object.

Example

Listing 9.2105 shows an example of the constructor property.

Listing 9.105 *Example of the constructor Property*

```
<server>

// Open a log file.
var myLog = new File("/data/logs/today.log");

// Open the log file for reading.
myLog.open("r");

if(myLog.constructor == File){
    write("Object created");
}

// Close the file.
myLog.close();

</server>
```

File.eof()

ES2+

Syntax

```
file.eof()
```

Description

The `eof()` method of the `File` object returns `true` if the position of the pointer within the file is past the end of the file. It returns `false` otherwise.

Example

Listing 9.106 reads a file and writes its contents to the page until the end of the file is found with the `eof()` method.

Listing 9.106 Reading a File Until You Come to the End of It, Which Can Be Evaluated Using the `eof()` Method

```

<server>

// Open a log file for reading.
var myLog = new File("/data/logs/today.log");
myLog.open("r");

// Write the contents of the log file to the page.
while (!myLog.eof()){
    myBytes = File.toString(myLog.readByte());
    write(myBytes);
}

// Close the file.
myLog.close();

</server>

```

File.error()

ES2+

Syntax

```
file.error()
```

Description

The `error()` method of the `File` object returns the operating system error code when an error occurs opening a file. This method returns `0` if there is no error, and `-1` if the file you invoke the method on is unable to be opened.

Example

Listing 9.107 opens a file for reading. If there was a problem during this operation, the error is written to the user's page.

Listing 9.107 Using the `error()` Method to Access an Error to Write to the User's Page

```

<server>

// Open a log file.
var myLog = new File("/data/logs/today.log");

// Open the log file for reading.
myLog.open("r");

```

Listing 9.107 Continued

```
if (myLog.error() == 0) {  
  
    // Perform actions on file.  
  
}else{  
  
    // Write out the error.  
    write('Error: ' + myLog.error());  
  
    // Clear the error.  
    myLog.clearError()  
}  
  
// Close the file.  
myLog.close();  
  
</server>
```

File.exists()

ES2+

Syntax

```
file.exists()
```

Description

The `exists()` method of the `File` object returns a boolean value based on the existence of the file in which it was invoked. If the file exists, the method returns `true`. It returns `false` if the file does not exist.

Example

Listing 9.108 opens a file and then checks to see whether it exists.

Listing 9.108 Using the `exists()` Method to See Whether a File Exists

```
<server>  
  
// Open a log file.  
var myLog = new File("/data/logs/today.log");  
  
// See if the file exists.  
if(myLog.exists()){  
    write('The file exists');  
}else{  
    write('The file does not exist');  
}  
  
</server>
```


File.flush()

ES2+

Syntax

```
file.flush()
```

Description

The `flush()` method of the `File` object is used to write buffered information to a file. This information is placed in a buffer when the `write()`, `writeln()`, and `writeByte()` methods are used. Note that this is not the same as the top-level `flush` function.

Example

Listing 9.109 opens a file for reading and another file for writing. If the file for reading exists, a string is written to the other file. The `flush()` method is used to write the buffered information to the file.

Listing 9.109 *Using the `flush()` Method*

```
<server>
```

```
// Open a log file.
var myLog = new File("/data/logs/today.log");
var mySummary = new File("/data/logs/summary.log");
```

```
myLog.open("r");
mySummary.open("w");
```

```
// See if the file exists.
if(myLog.exists()){
    mySummary.write('The file exists');
}else{
    mySummary.write('The file does not exist');
}
```

```
// Write the data in the buffer to the file.
mySummary.flush();
```

```
// Close the file.
myLog.close();
mySummary.close();
```

```
</server>
```

File.getLength()

ES2+

Syntax

```
file.getLength()
```

Description

The `getLength()` method of the `File` object returns the number of characters in a text file or the number of bytes in a binary file. If the method is unsuccessful, `-1` is returned.

Example

Listing 9.110 opens a file for reading and another file for writing. The `getLength()` method is used in a `for` loop to determine when to stop reading from the file.

Listing 9.110 Using the `getLength()` Method

```
<server>
```

```
// Open the files.
var myLog = new File("/data/logs/today.log");
var mySummary = new File("/data/logs/summary.log");
myLog.open("r");
mySummary.open("w");

// Write the contents of the log file to the page.
for(var i = 0; i <= myLog.getLength(); i++){
    myBytes = File.byteToString(myLog.readByte());
    mySummary.write(myBytes);
}

mySummary.flush();

// Close the files.
myLog.close();
mySummary.close();
```

```
</server>
```

File.getPosition()

ES2+

Syntax

```
file.getPosition()
```

Description

The `getPosition()` method of the `File` object returns the zero-based index position of the current pointer in the file. If the pointer is on the first character, `0` is returned. If there is an error, `-1` is returned.

Example

Listing 9.111 loops through each character, printing it on a new line next to its indexed location.

Listing 9.111 Using the `getPosition()` Method

```
<server>
```

```
// Open the files.
var myLog = new File("/data/logs/today.log");
var mySummary = new File("/data/logs/summary.log");
myLog.open("r");
mySummary.open("w");

// Write the contents of the log file to the page.
for(var i = 0; i <= myLog.getLength(); i++){
    myBytes = File.toString(myLog.readByte());
    mySummary.write('Character '+mySummary.getPosition()+ ' is '+myBytes+'<br>');
}

mySummary.flush();

// Close the files.
myLog.close();
mySummary.close();
```

```
</server>
```

File.open()

ES2+

Syntax

```
file.open(option)
```

Description

The `open()` method of the `File` object is used to open a file to read, write, and/or append to. The method returns `true` if it is successful and `false` otherwise. The options passed determine the mode in which the file is opened. The options are specified in Table 9.18.

Table 9.18 *Options of the open() Method*

Option	Description
a	This option opens a file for appending. If the file does not exist, it is created. This method always returns <code>true</code> .
a+	This option opens a file for reading and appending. If the file does not exist, it is created. This method always returns <code>true</code> .
r	This option opens a file for reading. If the file exists, the method returns <code>true</code> ; otherwise, it returns <code>false</code> .
r+	This option opens a file for reading and writing. If the file exists, the method returns <code>true</code> ; otherwise, it returns <code>false</code> . Reading and writing start at the beginning of the file.
w	This option opens a file for writing. If the file does not exist, it is created. If it does exist, it is overwritten. This method always returns <code>true</code> .
w+	This option opens a file for reading and writing. If the file does not exist, it is created. If it does exist, it is overwritten. This method always returns <code>true</code> .
optionb	Appending <code>b</code> to the end of any of these options specifies that you want to perform the operation in binary mode.

Example

Listing 9.112 shows how to open a file in read mode.

Listing 9.112 *Using the open() Method to Open a File*

```
<server>

// Initialize a file
var myLog = new File("/data/logs/today.log");

// Open the file in read mode
myLog.open("r");

// Close the file
myLog.close();

</server>
```

File.prototype
ES3+

Syntax

```
file.prototype.method = name
file.prototype.property = value
```

Description

The prototype property of the `File` object allows you to add methods and properties to the `File` object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.113 creates a new property and method of the `File` object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect (which it is), an error message is written to the page.

Listing 9.113 *Using the prototype Property to Create a New Property and Method*

```
<server>

// Define the method that we prototyped.
function verifyTEXT(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "text"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the connection object.
File.prototype.type = null;
File.prototype.isText = verifyTEXT;

// Initialize a file
var myLog = new File("/data/logs/today.log");

// Open the file in read mode
myLog.open("r");

// Using the prototype we defined, assign the type property.
myLog.type = "text";

// Check the type of the connection to see if it is valid
if(myLog.isText()){
    write(myLog + " has a valid type of " + myLog.type);
}else{
    write(myLog + " has an invalid type of " + myLog.type);
}

</server>
```

File.read()

ES2+

Syntax

```
file.read(num)
```

Description

The `read()` method of the `File` object starts at the current pointer in the file and reads *num* characters in the file. If you try to read characters past the end of the file, the method will read all the characters and stop. Use the `readByte()` method if you are trying to read the byte data.

Example

Listing 9.114 reads information from a text file and writes every other character to the user's page. The `setPosition()` method is used to move the pointer correctly, and the `read()` method is used to read the data.

Listing 9.114 *Using the read() Method*

```
<server>

// Open the files.
var myLog = new File("/data/logs/today.log");
myLog.open("r");

// Write some characters to the page.
for(var i = 1; i <= myLog.getLength(); i + 2){

    // Set the position of the pointer.
    myLog.setPosition(i);

    // Write every other character to the page.
    write(myLog.read(1));
}

// Close the file.
myLog.close();

</server>
```

File.readByte()

ES2+

Syntax

```
file.readByte(num)
```

Description

The `readByte()` method of the `File` object starts at the current pointer in the file and reads *num* bytes in the file. If you try to read bytes past the end of the file, the method will read all the bytes and stop. Use the `read()` method if you are trying to read the characters in text data.

Example

Listing 9.115 reads information from a text file and writes every other byte to the user's page. The `setPosition()` method is used to move the pointer correctly, and the `readByte()` method is used to read the data.

Listing 9.115 *Using the `readByte()` Method*

```
<server>

// Open the files.
var myLog = new File("/data/logs/today.dat");
myLog.open("rb");

// Write some characters to the page.
for(var i = 1; i <= myLog.getLength(); i + 2){

    // Set the position of the pointer.
    myLog.setPosition(i);

    // Write every other byte to the page.
    write(myLog.readByte(1));
}

// Close the file.
myLog.close();

</server>
```

File.readln()

ES2+

Syntax

```
file.readln()
```

Description

The `readln()` method of the `File` object starts at the current pointer position and reads the rest of the line. When the method is complete, it will return the pointer to the first character on the next line.

NOTE

The return (`\r`) and newline (`\n`) characters are not contained in the string this method returns. However, the newline character determines when the end of the line is reached.

Example

Listing 9.116 reads in the first line of a file and writes it to the user's page.

Listing 9.116 Using the `readln()` Method

```
<server>

// Open the files.
var myLog = new File("/data/logs/today.log");
myLog.open("r");

// Write the first line.
write(myLog.readln());

// Close the file.
myLog.close();

</server> >
```

File.setPosition()

ES2+

Syntax

```
file.setPosition(num)
file.setPosition(num, refPoint)
```

Description

The `setPosition()` method of the `File` object sets the pointer's position to a relative *num* location in the file. By default, this is relative to the beginning of the file, but you can pass a reference point to determine where this relative location is located. The possible values of this reference point are listed in Table 9.19.

Table 9.19 Possible Values of the Reference Point

Value	Description
0	Sets the pointer relative to the beginning of the file
1	Sets the pointer relative to the current pointer position
2	Sets the pointer relative to the end of the file

Example

Listing 9.117 uses the `setPosition()` method to access every other character in the file.

Listing 9.117 *Using the `setPosition()` Method*

```
<server>

// Open the files.
var myLog = new File("/data/logs/today.log");
myLog.open("r");

// Write some characters to the page.
for(var i = 1; i <= myLog.getLength(); i + 2){

    // Set the position of the pointer.
    myLog.setPosition(i);

    // Write every other character to the page.
    write(myLog.read(1));
}

// Close the file.
myLog.close();

</server>
```

File.toStringByte()

ES2+

Syntax

File.toStringByte(string)

Description

The `toStringByte()` method of the `File` object is used to convert the first character of the string passed to its binary equivalent.

Example

Listing 9.118 opens two files, one for reading and the other for appending. Strings are then read using the `read()` method from the first file, converted back to byte characters using the `toStringByte()` method, and then written to the second file. Both files are closed when the process has after completed.

Listing 9.118 *Using the `stringToByte()` Method to Convert the Strings Read into Bytes*

```
<server>

// Open a log file and a summary file.
var myLog = new File("/data/logs/today.dat");
var mySummary = new File("/data/logs/summary.dat");

// Open the log file for reading and the summary file for
// appending.
myLog.open("rb");
mySummary.open("ab");

// Append the contents of the log file to the summary file.
for(var i = 0; i <= myLog.getLength(); i++){
    myLog.setPosition(i);
    myByte = File.stringToBytes(myLog.read(1));
    mySummary.writeByte(myByte);
}

// Flush the buffer to the file.
mySummary.flush();

// Close the files.
myLog.close();
mySummary.close();

</server>
```

File.unwatch()

ES3+

Syntax

file.unwatch(property)

Description

The `unwatch()` method of the `File` object is used to turn off the watch for a particular *property*.

Example

Listing 9.119 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.119 Example of the `unwatch()` Method

```

<server>

// Define the method that we prototyped.
function verifyTEXT(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "text"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the connection object.
File.prototype.type = null;
File.prototype.isText = verifyTEXT;

// Initialize a file.
var myLog = new File("/data/logs/today.log");

// Open the file in read mode.
myLog.open("r");

// Using the prototype we defined, assign the type property.
myLog.type = "text";

// watch property
myLog.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myLog.isText()){
    write(myLog + " has a valid type of " + myLog.type);
}else{
    write(myLog + " has an invalid type of " + myLog.type);
}

// change value
myLog.type = null;

// turn off watch
myLog.unwatch("type");

```

Listing 9.119 Continued

```
// change value again
myLog.type = "ASCII";

</server>
```

File.watch()

ES3+

Syntax

```
file.watch(property, function)
```

Description

The `watch()` method of the `File` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.120 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.120 Example of the `watch()` Method

```
<server>

// Define the method that we prototyped.
function verifyTEXT(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "text"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the connection object.
File.prototype.type = null;
File.prototype.isText = verifyTEXT;
```

```
// Initialize a file.
var myLog = new File("/data/logs/today.log");

// Open the file in read mode.
myLog.open("r");

// Using the prototype we defined, assign the type property.
myLog.type = "text";

// watch property
myLog.watch("type", alertme);

// Check the type of the connection to see if it is valid.
if(myLog.isText()){
  write(myLog + " has a valid type of " + myLog.type);
}else{
  write(myLog + " has an invalid type of " + myLog.type);
}

// change value
myLog.type = null;

</server>
```

File.write()

ES2+

Syntax

```
file.write(string)
```

Description

The `write()` method of the `File` object writes the string passed to the method to the file on which it was invoked. The method returns `true` if it was successful and `false` otherwise. You should use the `read()` or `readln()` methods to read any string information from other files that you want to write. If you need to write binary information, use the `writeByte()` method.

Example

Listing 9.121 writes "Hello, World!" to the summary file opened.

Listing 9.121 Using `write()` to Write a String to a File

```
<server>
```

```
// Set an instance of a File.
var mySummary = new File("/data/logs/summary.log");
```

Listing 9.121 Continued

```
// Open the log file for writing.
mySummary.open("w");

// Write a string to the file.
mySummary.write('Hello, World!');

// Flush the buffer to the file.
mySummary.flush();

// Close the file.
mySummary.close();

</server>
```

File.writeByte()

ES2+

Syntax

```
file.writeByte(byte)
```

Description

The `writeByte()` method of the `File` object writes the byte passed to the method to the file on which it was invoked. The method returns `true` if it was successful and `false` otherwise. You should use the `readByte()` method to read any byte information from other files that you want to write. If you need to write text information, use the `write()` method.

Example

Listing 9.122 reads data from a binary file and writes it to another file.

Listing 9.122 Using `writeByte()` to Write Binary Data to a File

```
<server>

// Open a log file and a summary file.
var myLog = new File("/data/logs/today.dat");
var mySummary = new File("/data/logs/summary.dat");

// Open the log file for reading and the summary file for
// appending.
myLog.open("br");
mySummary.open("ba");

// Append the contents of the log file to the summary file.
for(var i = 0; i <= myLog.getLength(); i++){
    myLog.setPosition(i);
```

```
    myByte = myLog.readByte(1);
    mySummary.writeByte(myByte);
}

// Flush the buffer to the file.
mySummary.flush();

// Close the files.
myLog.close();
mySummary.close();

</server>
```

File.writeln()

ES2+

Syntax

```
file.writeln(string)
```

Description

The `writeln()` method of the `File` object writes the string passed to the method, followed by a carriage return to the file on which it was invoked. The method returns `true` if it was successful and `false` otherwise. You should use the `read()` or `readln()` methods to read any string information from other files you want to write. If you need to write binary information, use the `writeByte()` method.

Example

Listing 9.123 writes the line "Hello, World!" to the summary file opened.

Listing 9.123 Using `writeln()` to Write a String and Carriage Return to a File

```
<server>

// Set an instance of a File.
var mySummary = new File("/data/logs/summary.log");

// Open the log file for writing.
mySummary.open("w");

// Write a string to the file.
mySummary.writeln('Hello, World!');

// Flush the buffer to the file.
mySummary.flush();
```

Listing 9.123 Continued

```
// Close the file.  
mySummary.close();  
  
</server>
```

flush()

ES2+

Syntax

```
flush()
```

Description

The `flush()` function is a top-level function that is not associated with any object. The JavaScript runtime buffers the HTML page it is constructing in memory, and then sends it to the client after 64KB has been obtained. This function can be used to send this data on demand, which is useful when done before large database queries. Note this is not the same flush as the `File.flush()` method.

NOTE

The `flush()` function updates the cookie file as part of the HTTP header. If you are using a client cookie to maintain the `client` object, you should make any changes to this object before calling the `flush()` function.

Example

Listing 9.124 uses the `flush()` function to flush the buffer to the page after the initial write.

Listing 9.124 Using the `flush()` Function to Flush the Buffer to the Page

```
<server>  
  
// Write a string to the page.  
write('Hello, World!');  
  
// Flush the buffer to the client.  
flush();  
  
</server>
```


getOptionValue()

Nes2+

Syntax

```
getOptionValue(key, num)
```

Description

The `getOptionValue()` function is a top-level function that is not associated with any object. This function returns the same value as the client-side `Option.text` property. The `key` passed in as an argument represents the `name` attribute of the `<select>` tag, whereas the `num` parameter is the indexed position of the selected option. Specifying `num` to `0` can reference the first selected option, which might not be the first option.

Example

Listing 9.125 displays a multi-select option box that allows a user to select different sports. When the Submit button is clicked, the form is sent back to itself and the script then uses a `for` loop and the `getOptionValue()` function to write the submitted selections to the user's page.

Listing 9.125 Using the `getOptionValue()` Function

```
<html>
<head>
  <title> Using the getOptionValue Function</title>
</head>
<body>
<server>

// See if they have submitted or just need the form.
if(request.method == "POST"){

  // Store the number of selected options in a variable.
  var counter = getOptionValueCount("sports");

  // Write the title.
  write('<b>You selected the following options</b><hr size=1>');

  // Iterate through the options and write which ones were selected.
  for(var i = 0; i < counter; i++){

    // Get the Option.text values of the selected options.
    var optionValue = getOptionValue("sports", i)
    write('Option ' + i + ': ' + optionValue + '<br>')
  }
}else{
  // If this page was called and a form was not submitted, then write the
  // form to the page for the user to use.
```

Listing 9.125 Continued

```
write('<form name="myForm" method="post">');
write('<select name="sports" multiple size="4">');
write('<option>Baseball</option>');
write('<option>Football</option>');
write('<option>Basketball</option>');
write('<option selected>Soccer</option>');
write('<option>Rugby</option>');
write('</select>');
write('<input type="submit" value="Submit">');
write('</form>');
}

</server>
</body>
</html>
```

getOptionValueCount()

ES2+

Syntax

`getOptionValueCount(key)`

Description

The `getOptionValueCount()` function is a top-level function that is not associated with any object. This function returns the number of selected options passed to the function. The *key* parameter represents the name attribute of the `<select>` tag on which you want to invoke the function.

Example

Listing 9.126 shows how to use the `getOptionValueCount()` function to determine how many options were selected in a "sports" select box.

Listing 9.126 Using the `getOptionValueCount()` Function

```
<server>

// Store the number of selected options in a variable.
var counter = getOptionValueCount("sports");

</server>
```

Lock()

ES3+

Syntax

`new Lock()`

Description

A `Lock()` object is used when you enter a section of code that can only be accessed from a single user at a time. Not implementing an instance of this object when needed can cause you to run out of system resources and will generate a runtime error. This object has only one property, which is the prototype property.

This property can be used to create new methods and properties for the `Lock()` object. Table 9.20 contains the methods accessible from this object.

Table 9.20 *Methods of the `Lock()` Object*

Method	Description
<code>isValid()</code>	Verifies the construction of the <code>Lock()</code> object instance
<code>lock()</code>	Locks the code
<code>unlock()</code>	Unlocks the code
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property

Example

Listing 9.127 shows how you would create a new `Lock()` object instance. See the examples for `Lock.lock()` and `Lock.unlock()` for examples on locking and unlocking your code segments.

Listing 9.127 *Creating a New `Lock()` Object*

```
<server>

// Create a new Lock object.
var myLock = new Lock();

</server>
```

Lock.constructor

ES2+

Syntax

`lock.constructor`

Description

The constructor property of the `Lock` object specifies the function that creates the object.

Example

Listing 9.128 shows an example of the constructor property.

Listing 9.128 *The constructor **Property***

```
<server>

// Create a new Lock object.
var myLock = new Lock();

if(myLock.constructor == Lock){
    write("Object created");
}

</server>
```

Lock.isValid()

ES3+

Syntax

```
lock.isValid()
```

Description

The `isValid()` method of the `Lock` object verifies whether the lock was properly constructed. The method returns `true` if the lock was constructed successfully and `false` otherwise.

Example

Listing 9.129 shows how you can check whether a lock was properly constructed with an `if` statement.

Listing 9.129 *Verifying a Lock with the `isValid()` Method*

```
<server>

// Create a new Lock object.
var myLock = new Lock();

// Verify the lock.
if(!myLock.isValid()){
    write('There has been an error constructing your lock');
}else{

    // Success: perform operations here.

}

</server>
```

Lock.lock()

ES3+

Syntax

```
lock.lock()
```

Description

The `lock()` method of the `Lock` object locks the code in which you are working until you perform an `unlock`. If the code is already locked, this method will wait until it can get the lock, a timeout occurs, or an error occurs.

Example

Listing 9.130 assumes that there is a `project` object with a counter property, `num`. Because you only want to increment the counter with each new user request, it must be locked.

Listing 9.130 *Using the `lock()` Method to Lock Your Code*

```
<server>
```

```
// Create a new Lock object under the project object.
var project.myLock = new Lock();

// Verify the lock.
if(!project.myLock.isValid()){
    write('There has been an error constructing your lock');
}else{

    // Lock the code and increment a project counter.
    project.myLock.lock();
    project.hitCount.num += 1;

    // Unlock the code.
    project.myLock.unlock();

}
```

```
</server>
```

Lock.prototype

ES3+

Syntax

```
lock.prototype.method = name
lock.prototype.property = value
```

Description

The prototype property of the Lock object allows you to add methods and properties to the Lock object. If you are adding a method, you set the instance equal to the *name* of the method you have defined.

Example

Listing 9.131 creates a new property and method of the Lock object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to of the page.

Listing 9.131 *Using the prototype Property to Create a New Property and Method*

```
<server>

// Define the method that we prototyped.
function verifyProject(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "project"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the Lock object.
Lock.prototype.type = null;
Lock.prototype.isProject = verifyProject;

// Create a new Lock object under the project object.
var project.myLock = new Lock();

// Using the prototype we defined, assign the type property.
project.myLock.type = "project";

// Check the type of the lock to see if it is valid.
if(project.myLock.isProject()){
    write(project.myLock + " has a valid type of " + project.myLock.type);
}else{
    write(project.myLock + " has an invalid type of " + project.myLock.type);
}

</server>
```

Lock.unlock()

ES3+

Syntax

```
lock.unlock()
```

Description

The `unlock()` method of the `Lock` object unlocks the code you have locked. This method returns `true` if the unlocking was successful and `false` otherwise.

Example

Listing 9.132 assumes that there is a `project` object with a counter property, `num`. Because you only want to increment the counter with each new user request, it must be locked. After the counter has been incremented, the project is unlocked.

Listing 9.132 *Using the `unlock()` Method to Lock Your application*

```
<server>
```

```
// Create a new Lock object under the project object.
var project.myLock = new Lock();

// Verify the lock.
if(!project.myLock.isValid()){
    write('There has been an error constructing your lock');
}else{

    // Lock the code and increment a project counter.
    project.myLock.lock();
    project.hitCount.num += 1;

    // Unlock the code.
    project.myLock.unlock();

}
```

```
</server>
```

Lock.unwatch()

ES3+

Syntax

```
lock.unwatch(property)
```

Description

The `unwatch()` method of the `Lock` object is used to turn off the watch for a particular *property*.

Example

Listing 9.133 shows how the `unwatch()` method is used to stop watching the user-defined property `p`.

Listing 9.133 *Example of the `unwatch()` Method*

```
<server>
```

```
// Define the method that we prototyped.
function verifyProject(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "project"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Lock object.
Lock.prototype.type = null;
Lock.prototype.isProject = verifyProject;

// Create a new Lock object under the project object.
var project.myLock = new Lock();

// Using the prototype we defined, assign the type property.
project.myLock.type = "project";

// watch property
project.myLock.watch("type", alertme);

// Check the type of the lock to see if it is valid.
if(project.myLock.isProject()){
    write(project.myLock + " has a valid type of " + project.myLock.type);
}else{
    write(project.myLock + " has an invalid type of " + project.myLock.type);
}

// change value
project.myLock.type = null;
```



```
// turn off watch
project.myLock.unwatch("type");

// change value again
project.myLock.type = null;

</server>
```

Lock.watch()

ES3+

Syntax

```
lock.watch(property, function)
```

Description

The `watch()` method of the `Lock` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.134 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.134 *Example of the watch() Method*

```
<server>

// Define the method that we prototyped.
function verifyProject(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "project"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + " ) changed from " + oldValue + " to " + newValue);
    return newValue;
}
```

Listing 9.134 Continued

```
// Create a new property and method of the Lock object.
Lock.prototype.type = null;
Lock.prototype.isProject = verifyProject;

// Create a new Lock object under the project object.
var project.myLock = new Lock();

// Using the prototype we defined, assign the type property.
project.myLock.type = "project";

// watch property
project.myLock.watch("type", alertme);

// Check the type of the lock to see if it is valid.
if(project.myLock.isProject()){
    write(project.myLock + " has a valid type of " + project.myLock.type);
}else{
    write(project.myLock + " has an invalid type of " + project.myLock.type);
}

// change value
project.myLock.type = null;

</server>
```

ObjectContext

ASP1+

Syntax

Core ASP environment object

Description

The ObjectContext object is a core ASP environment object. The events and methods of this object are used to commit or abort a transaction that has been initiated by a script. This object requires the <%@ Transaction = Required %> directive to execute properly within your scripts. Table 9.21 contains the events and methods of this object.

Table 9.21 Methods and events of the ObjectContext Object

Type	Item	Description
Event	OnTransactionAbort	Occurs if the transaction is aborted.
	OnTransactionCommit	Occurs after a transactional script's transaction commits.

Type	Item	Description
Method	<code>SetAbort()</code>	Declares that the transaction did not complete and the resources should not be updated.
	<code>SetComplete()</code>	Declares that the script is not aware of any reasons for the transaction to not complete. If all components participating in the transaction also call <code>SetComplete()</code> , the transaction will complete.

Example

Listing 9.135 assumes that the variable *processData* has been set to true or false previously and, based on its value, either processes the transaction or aborts it.

Listing 9.135 Using the *ObjectContext* Object

```
<%@ Transaction = Required %>
<script runat="server" type="text/jscript" language="JScript">

// Assume processData is a boolean variable defined previously.
if(processData){
    ObjectContext.SetComplete
    Response.Write("Processing data...");
}else{
    ObjectContext.SetAbort
    Response.Write("Error: ending transaction");
}
</script>
```

ObjectContext.OnTransactionAbort

ASP1+

Syntax

```
function onTransactionAbort(){
    code
}
```

Description

The `onTransactionAbort` event of the *ObjectContext* object occurs when a transaction is aborted. When this happens, IIS calls the `onTransactionAbort()` function, if one is defined, and executes *code*.

Example

In Listing 9.136 we have defined the `onTransactionAbort` function that will be executed if a transaction is aborted. In the case this event is fired, a message will be written to output.

Listing 9.136 Using the `onTransactionAbort` Event

```
<%@ Transaction = Required %>
<script runat="server" type="text/jscript" language="JScript">
function onTransactionAbort(){
    Response.Write("Transaction has been aborted");
}
</script>
```

ObjectContext.OnTransactionCommit

ASP1+

Syntax

```
function onTransactionCommit(){
    code
}
```

Description

The `onTransactionCommit` event of the `ObjectContext` object occurs when a transaction has completed. When this happens, IIS calls the `onTransactionCommit()` function, if one is defined, and executes *code*.

Example

In Listing 9.137 we have defined the `onTransactionCommit()` function that will be executed if a transaction is committed. In the case this event is fired, a message will be written to output.

Listing 9.137 Using the `onTransactionCommit` Event

```
<%@ Transaction = Required %>
<script runat="server" type="text/jscript" language="JScript">
function onTransactionCommit(){
    Response.Write("Transaction has completed");
}
</script>
```

ObjectContext.SetAbort()

ASP1+

Syntax

```
ObjectContext.SetAbort()
```

Description

The `SetAbort()` method of the `ObjectContext` object aborts a transaction.

Example

In Listing 9.138 we have defined the `onTransactionAbort()` function that will be executed when the transaction is aborted, which we will induce with the `SetAbort()` method. When fired, a message will be written to be output.

Listing 9.138 Using the `SetAbort()` Method

```
<%@ Transaction = Required %>
<script runat="server" type="text/jscript" language="JScript">

// Define the function.
function onTransactionAbort(){
    Response.Write("Transaction has completed");
}

// Create an abort, which will call the function.
ObjectContext.SetAbort();

</script>
```

ObjectContext.SetComplete()

ASP1+

Syntax

```
ObjectContext.SetComplete()
```

Description

The `SetComplete()` method of the `ObjectContext` object overrides any previous calls to the `SetAbort()` method.

Example

In Listing 9.139 we have defined the `onTransactionAbort()` function that will be executed when the transaction is aborted, which we will induce with the `SetAbort()` method. When fired, we will stop the abort before the message is written to output.

Listing 9.139 Using the `SetComplete()` Method

```
<%@ Transaction = Required %>
<script runat="server" type="text/jscript" language="JScript">

// Define the function.
function onTransactionAbort(){
    Response.Write("Transaction has be aborted");
}
```

Listing 9.139 Continued

```
// Create an abort, which will call the function.
ObjectContext.SetAbort();

// Override the abort.
ObjectContext.SetComplete();

</script>
```

project

ES2+

Syntax

Created by the server-side JavaScript runtime when the application starts.

Description

The project object is shared by all users accessing the application. This object is created when the application is started, and destroyed when it is stopped. The project object is often used to store global properties that need to be accessed or modified by all user sessions. A common use of this object might be a counter holding the number of users who have accessed your application.

The object itself has no default properties, but properties can be created simply by specifying a name and value for the property. The following is an example of how to create a property and assign it a value, which is the IP address of the client requesting the application.

```
var project.lastIP = request.ip;
```

Table 9.22 contains a list and a description of the methods of the project object.

Table 9.22 Methods of the project Object

Method	Description
lock()	Locks the code
unlock()	Unlocks the code
unwatch()	Turns off the watch for a particular property
watch()	Turns on the watch for a particular property

Example

Listing 9.140 shows how you can create and increment a counter property of the project object. Before this counter is incremented, the project is locked and then is unlocked afterward.

Listing 9.140 Using the project Object

```
<server>

// Lock the code and increment a project counter.
project.lock();
project.hitCount += 1;

// Unlock the code.
project.unlock();

</server>
```

project.lock()

ES2+

Syntax

```
project.lock()
```

Description

The `lock()` method of the `project` object locks the code in which you are working until you perform an `unlock`. If the code is already locked, this method will wait until it can get the lock, a timeout occurs, or an error occurs.

Example

Listing 9.141 sets a `project` object with a counter property, `hitCount`. Because you only want to increment the counter with each new user request, it must be locked.

Listing 9.141 Using the `lock()` Method to Lock Your Code

```
<server>

// Lock the code and increment a project counter.
project.lock();
project.hitCount += 1;

// Unlock the application.
project.unlock();

</server>
```

project.unlock()

ES2+

Syntax

```
project.unlock()
```

Description

The `unlock()` method of the `project` object unlocks the code that you have locked. This method returns `true` if the unlocking was successful and `false` otherwise.

Example

Listing 9.142 sets a `project` object with a counter property, `hitCount`. Because you only want to increment the counter with each new user request, it must be locked. After the counter has been incremented, the project is unlocked.

Listing 9.142 Using the `unlock()` Method to Unlock Your Code

```
<server>

// Lock the code and increment a project counter.
project.lock();
project.hitCount += 1;

// Unlock the code.
project.unlock();

</server>
```

`project.unwatch()`

ES3+

Syntax

```
project.unwatch(property)
```

Description

The `unwatch()` method of the `project` object is used to turn off the watch for a particular *property*.

Example

Listing 9.143 shows how the `unwatch()` method is used to stop watching the user-defined property `p`.

Listing 9.143 Example of the `unwatch()` Method

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + " ) changed from " + oldValue + " to " + newValue);
    return newValue;
}
```



```

project.type = "work";

// watch property
project.watch("type", alertme);

// change value
project.type = null;

// turn off watch
project.unwatch("type");

// change value again
project.type = null;

</server>

```

project.watch()

ES3+

Syntax

```
project.watch(property, function)
```

Description

The `watch()` method of the `project` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.144 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.144 *Example of the watch() Method*

```

<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

project.type = "work";

// watch property
project.watch("type", alertme);

```

Listing 9.144 Continued

```
// change value
project.type = null;

</server>
```

redirect()

ES2+

Syntax

```
redirect(URL)
```

Description

The `redirect()` function is a top-level function that is not associated with any core object. This function is used to redirect the requesting browser to the URL it was passed as a parameter.

 NOTE

Use the `addClient()` function to preserve `client` object properties and their values.

Example

Listing 9.145 checks whether the browser asking for the page is Internet Explorer. The `redirect()` function is then used to redirect the browser accordingly.

Listing 9.145 Using the `redirect()` Function to Redirect a User's Browser

```
<server>

// Check to see if the browser is Internet Explorer.
if(request.agent.indexOf('MSIE') != -1){
    redirect(addClient("/iepages/index.html"));
}

// Redirect to another page if it is not IE.
}else{
    redirect(addClient("/defaultpages/index.html"));
}

</server>
```

registerCFunction()

ES2+

Syntax

```
registerCFunction(JSFuncName, libPath, CFunc)
```

Description

`registerCFunction()` is a top-level function that is not associated with any core object. `RegisterCFunction()` is used to register a C function in a shared library as a JavaScript function so it can be used in your scripts. These libraries are the pre-built .dll files on Windows machines and .so files on Unix machines. `RegisterCFunction()` takes the JavaScript name you want to assign the C function, the path to the library, and the actual name of the C function you want to register in the library. The function returns a boolean value based on the success of the registration.

After you call this function, you can use the C library with the server-side JavaScript `callC()` function. `callC()` takes the JavaScript name you assigned and any parameters that need to be passed to the function.

Example

Listing 9.146 registers an external C library, `extlib.dll` that contains a function named `getMyDate`. The registration of this function assigns the name `JSExtLib` to be used within the script. If the function successfully registers, the `callC` function is used to call the C function and pass it two parameters. The results are written to the user's page. If the function does not register properly, an error is written to the user's page.

Listing 9.146 Registering a C Function in a Shared Library with

`registerCFunction()`

```
<server>

// Register the library and function, assigning it a JavaScript
// function name.
var myExternalLib = registerCFunction("JSExtLib", "c:/winnt/extlib.dll",
  ➤ "getMyDate")

// If the library registered without error, then call it using the
// callC function. If it failed, then write an error to the page.
if (myExternalLib) {
    write(callC("getMyDate", 1999, 2000));
}else{
    write("There was an error processing this external library function");
}

</server>
```

Request

ASP1+

Syntax

Core ASP environment object

Description

The Request object is a core ASP environment object. This object provides a means to retrieve HTTP information passed to the server from the user agent during a request. Table 9.23 contains the collections, methods, and a property of this object.

Table 9.23 *Collections, Method, and a Property of the Request Object*

Type	Item	Description
Collection	ClientCertificate	Values of fields stored in the client certificate.
	Cookies	Values of cookies.
	Form	Values of form elements.
	QueryString	Values of variables in the HTTP query string.
	ServerVariables	Values of predetermined environment variables.
Method	BinaryRead()	Retrieves POST data sent to the server from the client.
Property	TotalBytes	Read-only property that specifies the total number of bytes the client is sending in the body of the request.

Example

Listing 9.147 shows how you can use the Request object to access a specific querystring key, and then write out its value.

Listing 9.147 *Using the Request Object*

```
<script runat="server" type="text/jscript" language="JScript">
```

```
// Write out the value of the "name" querystring entry.
```

```
Response.Write(Request.QueryString("name"));
```

```
</script>
```

Request.BinaryRead()

ASP1+

Syntax

Request.BinaryRead(*num*)

Description

The `BinaryRead()` method of the `Request` object retrieves data sent to the server in a POST request, and stores the information in an array of unsigned bytes, called a *SafeArray*. The *num* represents the number of bytes to be read into the array.

Example

Listing 9.148 shows how you can check the length of a POST, and then read the data into a *SafeArray*.

Listing 9.148 Using the `BinaryRead()` Method

```
<script runat="server" type="text/jscript" language="JScript">

// Check the length of the POST.
var myLen = Request.TotalBytes;

// Read the data in.
var mySafeArray = Request.BinaryRead(myLen);

</script>
```

Request.ClientCertificate

ASP1+

Syntax

```
Request.ClientCertificate(field)
Request.ClientCertificate(subfield)
```

Description

The `ClientCertificate` collection of the `Request` object retrieves the certification fields from a request. If the request uses the SSL3.0/PCT1 protocol (also referred to as HTTPS) to connect to a server, the server will request certification at which time the browser sends the certification fields. If no certificate is sent, the `ClientCertificate` collection returns `EMPTY`.

Table 9.24 contains a list of *fields* that can be passed to this collection, whereas Table 9.25 contains a list of possible *subfields*.

Table 9.24 Fields of the `ClientCertificate` Collection

Field	Description
Certificate	String containing the binary stream of the entire certificate in ASN.1 format.
Flags	Set of flags providing additional client certificate information. These flags can be <code>ceCertPresent</code> (client certificate is present) or <code>ceUnrecognizedIssuer</code> (last certification in this chain is from an unknown issuer).

Table 9.24 Continued

Field	Description
Issuer	String containing a list of <i>subfield</i> values containing information about the issuer of the certificate. If this value is specified without a <i>subfield</i> , the collection returns a comma-separated list of <i>subfields</i> (that is, C=US, O=Verisign, and so on).
SerialNumber	String containing the certification serial number as an ASCII representation of hexadecimal bytes separated by hyphens (-).
Subject	String containing a list of <i>subfield</i> values. If this value is specified without a <i>subfield</i> , the collection returns a comma-separated list of <i>subfields</i> (that is, C=US, O=Verisign, and so on).
ValidFrom	Date specifying when the certificate becomes valid.
ValidUntil	Date specifying when the certificate expires. The year value is displayed as a four-digit number.

NOTE

To use the Flags, you must include the client-certificate include file (cerjavas.inc) in your ASP page. These files are installed in the \Inetpub\ASPSamp\Samples directory.

Table 9.25 Subfields of the ClientCertificate Collection

Subfield	Description
C	Specifies the name of the country/region of origin.
CN	Specifies the common name of the user. (This is only used with the Subject field.)
GN	Specifies a given name.
I	Specifies a set of initials.
L	Specifies a locality.
O	Specifies the company or organization name.
OU	Specifies the name of the organizational unit.
S	Specifies a state or province.
T	Specifies the title of the person or organization.

The syntax for the use of *subfields* can be a bit strange looking, so here is a quick example. Notice that the *subfield* is actually the result of concatenating the field in our previous table with the *subfield* identifier.

```
Request.ClientCertificate("SubjectCN");
```

Example

In Listing 9.149 we write out the Subject field of a client certificate present in a request.

Listing 9.149 Using the ClientCertificate Collection

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(Request.ClientCertificate("Subject"));
</script>
```

Request.Cookies

ASP1+

Syntax

```
Request.Cookies(name)
Request.Cookies(name).HasKeys
Request.Cookies(name)(subkey)
```

Description

The Cookies collection of the Request object provides access to any cookies passed to the server. You can retrieve the value of the cookie by passing the *name* of the cookie. Additionally, you can pass the HasKeys attribute to check if the collection is a cookie dictionary. If so, you can pass the *subkey* property to retrieve any subkey values from cookie dictionaries.

Example

Listing 9.150 shows how you can use the AppendToLog() method in conjunction with the Request.Cookies collection to access and write the value of a cookie named ID to the end of the log file.

Listing 9.150 Using the Cookies Collection

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(Request.Cookies("ID"));
Response.Flush();
</script>
```

Request.Form

ASP1+

Syntax

```
Request.Form(name)
Request.Form(name)(num)
```

Description

The Form collection of the Request object retrieves the values of form elements that have been POSTed to the server. This collection is used by passing the *name* of the element you want to retrieve. If multiple values are passed for a given *name*, you can specify the *num* location of the value you want to retrieve.

NOTE

This collection can only hold 100KB of information. If your POST is larger than that, you should use the `Request.Binary()` method.

Example

Listing 9.151 shows how you can check the number of values for a given form name, and then write the last value to the page.

Listing 9.151 Using the *Form Collection*

```
<script runat="server" type="text/jscript" language="JScript">

// See how many values are in the "interest" element.
myElementLen = Request.Form("interest").Count;

// Write out the last value.
Response.Write(Request.Form("interest")(myElementLen));
</script>
```

Request.Form.Count

ASP1+

Syntax

`Request.Form(name).Count`

Description

The `Count` property of the `Form` collection of the `Request` object contains the total number of values for a given form element that has been POSTed to the server. This collection is used by passing the *name* of the element you want to retrieve a count for.

Example

Listing 9.152 shows how you can check the number of values for a given form name, and then write the last value to the page.

Listing 9.152 Using the *Count Property of the Form Collection*

```
<script runat="server" type="text/jscript" language="JScript">

// See how many values are in the "interest" element.
myElementLen = Request.Form("interest").Count;

// Write out the last value.
Response.Write(Request.Form("interest")(myElementLen));
</script>
```

Request.QueryString

ASP1+

Syntax

```
Request.QueryString(name)  
Request.QueryString(name)(num)
```

Description

The QueryString collection of the Request object retrieves the values of form elements that have been passed to the server using the GET method. This collection is used by passing the *name* of the element you want to retrieve. If multiple values are passed for a given *name*, you can specify the *num* location of the value you want to retrieve.

NOTE

A QueryString is signified as all characters after the first occurrence of the "?" character.

Example

Listing 9.153 shows how you can check the number of values for a given querystring name, and then write the last value to the page.

Listing 9.153 Using the QueryString Collection

```
<script runat="server" type="text/jscript" language="JScript">  
  
// See how many values are in the "interest" element.  
myElementLen = Request.QueryString("interest").Count;  
  
// Write out the last value.  
Response.Write(Request.QueryString("interest")(myElementLen));  
</script>
```

Request.QueryString.Count

ASP1+

Syntax

```
Request.QueryString(name).Count
```

Description

The Count property of the QueryString collection of the Request object contains the total number of values for a given form element that has been sent to the server in a GET request. This collection is used by passing the *name* of the element you want to retrieve a count for.

Example

Listing 9.154 shows how you can check the number of values for a given form element, and then write the last value to the page.

Listing 9.154 Using the *Count* Property of the *QueryString Collection*

```
<script runat="server" type="text/jscript" language="JScript">

// See how many values are in the "interest" element.
myElementLen = Request.QueryString("interest").Count;

// Write out the last value.
Response.Write(Request.QueryString("interest")(myElementLen));
</script>
```

Request.ServerVariables

ASP1+

Syntax

`Request.ServerVariables(name)`

Description

The `ServerVariables` collection of the `Request` object retrieves a list of predefined server environment variables. You can access the value of a given variable by passing the variable's *name*. Table 9.26 contains a list of possible values for *name*.

Table 9.26 Possible `ServerVariables` for Which Information Can Be Obtained

Variable	Description
ALL_HTTP	All HTTP headers sent by the client. Note that this method places an <code>HTTP_</code> prefix before the header name, and the header name is always capitalized.
ALL_RAW	Retrieves all headers in raw form.
APPL_MD_PATH	Retrieves the metabase path for the ISAPI application .dll.
APPL_PHYSICAL_PATH	Retrieves the physical path corresponding to the metabase path.
AUTH_PASSWORD	Value entered in the client's authentication dialog.
AUTH_TYPE	Authentication method used to validate users when they attempt to access a protected script.
AUTH_USER	Raw authenticated username.
CERT_COOKIE	Unique ID for client certificate.
CERT_FLAGS	bit0 is set to 1 if the client certificate is present. bit1 is set to 1 if the cCertification authority of the client certificate is invalid.

Variable	Description
CERT_ISSUER	Issuer field of the client certificate (O=MS, OU=IAS, CN=user name, C=USA).
CERT_KEYSIZE	Number of bits in Secure Sockets Layer connection key size.
CERT_SECRETKEYSIZE	Number of bits in server certificate private key.
CERT_SERIALNUMBER	Serial number field of the client certificate.
CERT_SERVER_ISSUER	Issuer field of the server certificate.
CERT_SERVER_SUBJECT	Subject field of the server certificate.
CERT_SUBJECT	Subject field of the client certificate.
CONTENT_LENGTH	Length of the content as given by the client.
CONTENT_TYPE	Data type of the content.
GATEWAY_INTERFACE	Revision of the CGI specification used by the server.
HTTP_<HeaderName>	Value stored in the header HeaderName.
HTTP_ACCEPT	Returns the value of the Accept header.
HTTP_ACCEPT_LANGUAGE	Returns a string describing the language to use for displaying content.
HTTP_USER_AGENT	Returns a string describing the browser that sent the request.
HTTP_COOKIE	Returns the cookie string that was included with the request.
HTTP_REFERER	Returns a string containing the URL of the page that referred the request, but does not include redirect requests.
HTTPS	Returns ON if request came in through SSL or returns OFF if the request is for a non-secure channel.
HTTPS_KEYSIZE	Number of bits in the Secure Sockets Layer connection key size.
HTTPS_SECRETKEYSIZE	Number of bits in the server certificate private key.
HTTPS_SERVER_ISSUER	Issuer field of the server certificate.
HTTPS_SERVER_SUBJECT	Subject field of the server certificate.
INSTANCE_ID	Textual format ID for the IIS instance.
INSTANCE_META_PATH	Metabase path for the instance of IIS.
LOCAL_ADDR	Returns the Server Address on which the request came in. This is important on multihomed machines where multiple IP addresses can be bound to the machine.
LOGON_USER	Windows account the user is logged into.
PATH_INFO	Extra path information as given by the client.
PATH_TRANSLATED	Translated version that takes the path and performs any necessary virtual-to-physical mapping.
QUERY_STRING	Query information stored in the string following the question mark (?).
REMOTE_ADDR	IP address of the remote host.

Table 9.26 *Continued*

Variable	Description
REMOTE_HOST	Name of the host making the request.
REMOTE_USER	Unmapped username string sent in by the user.
REQUEST_METHOD	Method used to make the request.
SCRIPT_NAME	Virtual path to the script being executed.
SERVER_NAME	Server's hostname, DNS alias, or IP address.
SERVER_PORT	Port number.
SERVER_PORT_SECURE	String containing 1 if the request is handled on the secure port; otherwise it will be 0.
SERVER_PROTOCOL	Name and revision of the request information protocol.
SERVER_SOFTWARE	Name and version of the server software that answers the request and runs the gateway.
URL	Gives the base portion of the URL.

Example

Listing 9.155 shows how you can write out the user agent string of the browser making the request.

Listing 9.155 *Using the* **ServerVariables Collection**

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(Request.ServerVariables("HTTP_USER_AGENT"));
</script>
```

Request.TotalBytes

ASP1+

Syntax

```
Request.TotalBytes
```

Description

The `TotalBytes` property of the `Request` object is a read-only property that contains the total number of bytes the user agent sent in the body of the request.

Example

Listing 9.156 shows how you can check the length of a POST, and then read the data into a *SafeArray*.

Listing 9.156 Using the `TotalBytes` Property

```

<script runat="server" type="text/jscript" language="JScript">

// Check the length of the POST.
var myLen = Request.TotalBytes;

// Read the data in.
var mySafeArray = Request.BinaryRead(myLen);

</script>

```

request

ES2+

Syntax

Core object created by JavaScript runtime for each client request.

Description

The request object contains specific information about the client's request. The object itself has several properties, as well as any properties that you might be passing from a form. This object also inherits `Object.watch()` and `Object.unwatch()`, which are defined in Chapter 7, "Core Language." The properties of this object are listed in Table 9.27.

Table 9.27 Properties of the request Object

Property	Description
<code>agent</code>	Contains the user-agent string sent in the HTTP request.
<code>imageX</code>	The x coordinate of an imagemap request.
<code>imageY</code>	The y coordinate of an imagemap request.
<code>formKey</code>	This could be any key, of a <code>key=value</code> pair, passed from within a <code><form></code> tag on a page.
<code>ip</code>	The IP address of the user request.
<code>method</code>	The HTTP method used in the request.
<code>protocol</code>	The protocol level the requesting agent supports.

Example

Listing 9.157 writes the various properties of a request to the page.

Listing 9.157 Using the request Object to See Properties of the Requesting Agent

```

<server>

// Write each of the properties of the request to the page.
write('User Agent: ' + request.agent + '<br>');
write('Y-coordinate of ISMAP: ' + request.imageY + '<br>');

```

Listing 9.157 Continued

```
write('X-coordinate of ISMAP: ' + request.imageX + '<br>');
write('IP Address: ' + request.ip + '<br>');
write('HTTP Method: ' + request.method + '<br>');
write('Protocol Version: ' + request.protocol);

</server>
```

request.agent

ES2+

Syntax

`request.agent`

Description

The `agent` property of the `request` object contains the user agent string of the requesting application. The following is an example of a user agent string for Navigator 4.5 on Windows 98.

```
Mozilla/4.5 (Win98; U)
```

You can use several methods of the `String` object in conjunction with this property to determine the type and version of the browser requesting the page.

Example

Listing 9.158 examines the requesting browser's agent string and redirects Internet Explorer browsers to an alternative page.

Listing 9.158 Accessing the *agent* Property of the *request* Object

```
<server>

// Check to see if the browser is Internet Explorer.
if(request.agent.indexOf('MSIE') != -1){
    redirect(addClient("/iepages/index.html"));
}

</server>
```

request.formKey

ES2+

Syntax

`request.formKey`

Description

The *formKey* property of the request object represents any form key that is sent. A key is specified by the name attribute in any element of an HTML `<form>`.

Example

Listing 9.159 shows how you can have a form with a “sports” key and use the `request.formKey` property to read the value.

Listing 9.159 Reading a *formKey* Property

```
<html>
<head>
  <title>Using the formKey Property</title>
</head>
<body>
<server>

// See if they have submitted or just need the form.
if(request.method == "POST"){

  // Print the selected option to the user's page.
  write('You selected ' + request.sports);

}else{
  // If this page was called and a form was not submitted, then write the
  // form to the page for the user to use.

  write('<form name="myForm" method="post">');
  write('<select name="sports">');
  write('<option>Baseball</option>');
  write('<option>Football</option>');
  write('<option>Basketball</option>');
  write('<option>Soccer</option>');
  write('<option>Rugby</option>');
  write('</select>');
  write('<input type="submit" value="Submit">');
  write('</form>');
}

</server>
</body>
</html>
```

request.imageX

ES2+

Syntax

`request.imageX`

Description

The `imageX` property of the request object contains the x coordinate of the imagemap request sent from the browser. These requests come in the following form, where *x* is the numeric x-coordinate and *y* is the numeric y coordinate:

`http://www.purejavascript.com/clickthru.html?x,y`

Example

Listing 9.160 pulls out the x and y coordinates passed in the request and writes them to the user's page.

Listing 9.160 Accessing the `imageX` Property

```
<server>

// Write the imageX and imageY properties to the page.
write('Y-coordinate of ISMAP: ' + request.imageY + '<br>');
write('X-coordinate of ISMAP: ' + request.imageX);

</server>
```

request.imageY

ES2+

Syntax

`request.imageY`

Description

The `imageY` property of the request object contains the y coordinate of the imagemap request sent from the browser. These requests come in the following form, where *x* is the numeric x coordinate and *y* is the numeric y coordinate:

`http://www.purejavascript.com/clickthru.html?x,y`

Example

Listing 9.161 pulls out the x and y coordinates passed in the request and writes them to the user's page.

Listing 9.161 Accessing the `imageY` Property

```
<server>

// Write the imageX and imageY properties to the page.
write('Y-coordinate of ISMAP: ' + request.imageY + '<br>');
write('X-coordinate of ISMAP: ' + request.imageX);

</server>
```

request.ip

ES2+

Syntax

`request.ip`

Description

The `ip` property of the `request` object specifies the IP address of the requesting client.

Example

Listing 9.162 checks the IP address of the requesting client. If the IP address is not within the correct domain, the browser is redirected to an “unauthorized IP” page.

Listing 9.162 Using the `ip` Property to Verify Authorization

```
<server>

// See if the IP address is ok.
if(request.ip.indexOf("207.200.75.") != -1){

    // Send them to an authorized page.
    redirect(addClient("/authorized.html"));

}else{

    // Send them to an error page.
    redirect(addClient("/unauthorized.html"));

}

</server>
```

request.method

ES2+

Syntax

`request.method`

Description

The `method` property of the `request` object specifies the HTTP method used in the request. This can either be `PUT` or `GET`, depending on the submission type.

Example

Listing 9.163 shows how you can use the same page to display a form and interpret it. If the method is `POST`, you know it was a form submission. If it was `GET`, you display the form.

Listing 9.163 Evaluating the method *Property*

```
<server>

// See if they have submitted or just need the form.
if(request.method == "POST"){

    // Perform your processing of the form here.

}else{

    // Write the actual HTML form here.

}

</server>
```

request.protocol

ES2+

Syntax`request.protocol`**Description**

The `protocol` property of the `request` object specifies the version of the HTTP protocol used in the request. This version is conveyed in the following format:

`HTTP/1.0`**Example**

Listing 9.164 takes a look at the `protocol` property to see whether the request is HTTP 1.1. You can then insert any processing code needed, based on the protocol version supported by the client.

Listing 9.164 Accessing the protocol *Property*

```
<server>

// Check the protocol version.
if(request.protocol.indexOf("1.1") != -1){

    // Perform your HTTP 1.1 processing of the form here.

}else{

    // Perform non-HTTP 1.1 processing here.
```

```
}  
  
</server>
```

request.unwatch()

ES3+

Syntax

```
request.unwatch(property)
```

Description

The `unwatch()` method of the `request` object is used to turn off the watch for a particular *property*.

Example

Listing 9.165 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.165 *Example of the* `unwatch()` *Method*

```
<server>  
  
// function that is called if property's value changes  
function alertme(id, oldValue, newValue){  
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);  
    return newValue;  
}  
  
// watch property  
request.watch("agent", alertme);  
  
// change value  
request.agent = null;  
  
// turn off watch  
request.unwatch("agent");  
  
// change value again  
request.agent = null;  
  
</server>
```

request.watch()

ES3+

Syntax

```
request.watch(property, function)
```

Description

The `watch()` method of the `request` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.166 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.166 *Example of the* `watch()` *Method*

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// watch property
request.watch("agent", alertme);

// change value
request.agent = null;

</server>
```

Response

ASP1+

Syntax

Core ASP environment object

Description

The `Response` object is a core ASP environment object. This object is used to send data back to the user agent making the request. Table 9.28 contains the list of collections, methods, and properties in this object.

Table 9.28 *Collections and Methods in the Response Object*

Type	Item	Description
Collection	Cookies	Specifies and sets cookie values.
Method	AddHeader()	Sets an HTML header name to a value.
	AppendToLog()	Adds a string to the end of the server log entry for this request.
	BinaryWrite()	Writes information without any character-set conversion.
	Clear()	Erases any buffered output.
	End()	Stops processing the .asp file and returns the current result.
	Flush()	Sends buffered output immediately.
	Redirect()	Sends a redirect message to the browser.
	Write()	Writes a variable to the current HTTP output as a string.

Example

In Listing 9.167 we use the `Write()` method of the `Response` object to write "Hello World" to the page.

Listing 9.167 Using the `Write()` Method of the `Response` Object

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write("Hello World");
</script>
```

Response.AddHeader()

ASP1+

Syntax

`Response.AddHeader(name, value)`

Description

The `AddHeader()` method of the `Response` object is used to add the *name* header, which is assigned *value*, to the HTTP header.

Example

In Listing 9.168 we show how you can add a P3P header pointing to your privacy policy reference file.

Listing 9.168 Using the AddHeader() Method

```
<script runat="server" type="text/jscript" language="JScript">
Response.AddHeader('P3P', 'policyref="http://www.purejavascript.com/p3p.xml"')
Response.Flush()
</script>
```

Response.AppendToLog()

ASP1+

Syntax

Response.AppendToLog(*string*)

Description

The AppendToLog() method/property of the Response object appends *string* to the end of the log entry for the given request. If called multiple times for a given request, it will continue appending to the end of that log entry.

Example

Listing 9.169 shows how you can use the AppendToLog() method in conjunction with the Request.Cookies collection to access and write the value of a cookie named ID to the end of the log file.

Listing 9.169 Using the AppendToLog() Method

```
<script runat="server" type="text/jscript" language="JScript">
Response.AppendToLog(Request.Cookies("ID"));
Response.Flush();
</script>
```

Response.BinaryWrite()

ASP1+

Syntax

Response.BinaryWrite(*data*)

Description

The BinaryWrite() method of the Response object is used to write non-string *data* back to the user agent. This parameter should be of type VT_ARRAY or VT_UI1, which is a variant array of unsigned one-byte characters.

Example

In Listing 9.170 we assume that you have an object that will create binary data, and then we use the BinaryWrite() method to write out the data.

Listing 9.170 Using the `BinaryWrite()` Method

```
<script runat="server" type="text/jscript" language="JScript">

// Assumes you have an object that will create binary data.
Set myBin = Server.CreateObject("myApp.createBin");
myResult = myBin.createGif;

// Send the data out.
Response.BinaryWrite(myResult);
</script>
```

Response.Buffer

ASP1+

Syntax

`Response.Buffer = boolean`

Description

The `Buffer` property of the `Response` object specifies whether the server should buffer page output or not. When *boolean* is `TRUE`, all data is buffered and is not sent to the user agent until the page has finished processing, or the `Response.End()` or `Response.Flush()` methods are called. Because this property cannot be set after data has been sent to the agent, this needs to be set in the first line of your file.

Example

Listing 9.171 shows how to tell the server to buffer the content, and then send it when the `Response.Flush()` method is called.

Listing 9.171 Using the `Buffer` Property

```
<script runat="server" type="text/jscript" language="JScript">
Response.Buffer = TRUE;
Response.Write("Hello World");
Response.Flush();
</script>
```

Response.CacheControl

ASP1+

Syntax

`Response.CacheControl = string`

Description

The `CacheControl` property of the `Response` object is used to override the `Private` default *string* by setting it to `Public`. The default setting states that only private

caches can cache the content generated by this page, whereas a `Public` setting informs proxies and other servers that they can cache the content.

Example

Listing 9.172 shows how you can use the `CacheControl` property to allow proxies to cache ASP content.

Listing 9.172 *Using the `CacheControl` Property*

```
<script runat="server" type="text/jscript" language="JScript">
Response.CacheControl = Public;
</script>
```

Response.Charset

ASP1+

Syntax

`Response.Charset = charset`

Description

The `Charset` property of the `Response` object appends the *charset* to the end of the content-type HTTP header directive.

Example

Listing 9.173 shows how you can append ISO-LATIN-7 to the end of the content-type is header.

Listing 9.173 *Using the `Charset` Property*

```
<script runat="server" type="text/jscript" language="JScript">

// Generates content-type:text/html; charset=ISO-LATIN-7.
Response.Charset = "ISO-LATIN-7";
</script>
```

Response.Clear()

ASP1+

Syntax

`Response.Clear()`

Description

The `Clear()` method of the `Response` object is used to erase any buffered HTTP response body output, but not HTTP response header output. For this method to work, you must first set `Response.Buffer = TRUE`.

Example

Listing 9.174 shows how you can use the `Clear()` method to clear any buffered HTTP response body data.

Listing 9.174 *Using the `Clear()` Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Turn on buffering.
Response.Buffer = TRUE;

// Write data to buffer then flush it.
Response.Write("Hello World");
Response.Flush();

// Clear response body.
Response.Clear()
</script>
```

Response.ContentType

ASP1+

Syntax

`Response.ContentType = contentType`

Description

The `ContentType` property of the `Response` object sets the content-type header directive. If no value is passed, `text/html` is assumed.

Example

Listing 9.175 shows how you can pass back your content as plain ASCII text, rather than HTML.

Listing 9.175 *Using the `ContentType` Property*

```
<script runat="server" type="text/jscript" language="JScript">
Response.ContentType = "text/plain"
Response.Write('<b>You will see the tags, and not bold text</b>');
</script>
```

Response.Cookies

ASP1+

Syntax

```
Response.Cookies(name) = value
Response.Cookies(name).attribute = value
Response.Cookies(name)(subkey) = value
```

Description

The Cookies collection of the Response object creates and stores any cookies to pass to the user agent. You can create a cookie called *name* of *value* using the first syntactical method. It is also possible to pass a *subkey*, which creates a cookie dictionary and inserts *value* as a key in that dictionary. Additionally, you can pass a specific *attribute* to just set or change part of the cookie.

Example

Listing 9.176 shows how you can use the Response.Cookie() collection to create, collect, and pass back a cookie with two subkeys.

Listing 9.176 Using the Cookies Collection

```
<script runat="server" type="text/jscript" language="JScript">

// Sends out Set-Cookie:ID=site=abc&network=123
Response.Cookies("ID")("site") = "abc"
Response.Cookies("ID")("network") = "123"
</script>
```

Response.End()

ASP1+

Syntax

Response.End()

Description

The End() method of the Response object tells the server to stop processing the script and return the current results.

Example

Listing 9.177 shows how you can call this method to stop execution of the current page.

Listing 9.177 Using the End() Method

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write("this will be written");
Response.End();
Response.Write("this will not be written");
</script>
```

Response.Expires

ASP1+

Syntax

Response.Expires = num

Description

The Expires property of the Response object sets the expiration header for the returned document by specifying the *num* of minutes before the document expires. Browsers use this to measure the amount of time they should cache the document.

Example

Listing 9.178 shows how you can set an expiration of only five minutes for the given page.

Listing 9.178 Using the Expires Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Expires = 5;  
Response.Write("Hello World");  
</script>
```

Response.ExpiresAbsolute

ASP1+

Syntax

Response.ExpiresAbsolute = *datetime*

Description

The ExpiresAbsolute property of the Response object sets the absolute expiration date for the returned document by specifying the *datetime* the document expires. Browsers use this to measure the amount of time they should cache the document.

Example

Listing 9.179 shows how you can set the expiration to October 31st, 2001 at 8:00 a.m. for the given page.

Listing 9.179 Using the ExpiresAbsolute Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.ExpiresAbsolute = 'October 31,2001 08:00:00';  
Response.Write("Hello World");  
</script>
```

Response.Flush()

ASP1+

Syntax

Response.Flush()

Description

The `Flush()` method of the `Response` object is used to send the currently buffered content to the user agent. For this method to work properly, the `Response.Buffer` property must be set to `TRUE`.

Example

Listing 9.180 shows how to tell the server to buffer the content, and then send it when the `Response.Flush()` method is called.

Listing 9.180 Using the `Flush()` Method

```
<script runat="server" type="text/jscript" language="JScript">
Response.Buffer = TRUE;
Response.Write("Hello World");
Response.Flush();
</script>
```

Response.IsClientConnected

ASP1+

Syntax

`Response.IsClientConnected`

Description

The `IsClientConnected` property of the `Response` object is a boolean value that lets you see whether a client is still connected.

Example

Listing 9.181 shows how you can evaluate the `IsClientConnected` property within an `if` statement before writing out your content. If the client is not connected, it writes an error to the Web log file.

Listing 9.181 Using the `IsClientConnected` Property

```
<script runat="server" type="text/jscript" language="JScript">
if(Response.IsClientConnected){
    Response.Write("Client is connected");
}else{
    Response.AppendToLog("Error: client disconnected");
}
</script>
```

Response.PICS

ASP1+

Syntax

`Response.PICS(label)`

Description

The PICS property of the Response object adds the PICS-label HTTP header directive to the current output. The *label* property contains a properly formatted PICS label.

Example

Listing 9.182 shows how you can use the PICS property to add the PICS-label HTTP header directive to the current output.

Listing 9.182 *Using the PICS Property*

```
<script runat="server" type="text/jscript" language="JScript">
Response.PICS("(PICS-1.1 <http://www.rsac.org/ratingv01.html> labels on "
               & chr(34) & "2001.01.05T08:15-0500" & chr(34) & " until "
               & chr(34) & "2002.12.31T23:59-0000" & chr(34)
               & " ratings (v 0 s 0 l 0 n 0))");
</script>
```

Response.Redirect()

ASP1+

Syntax

Response.Redirect(*url*)

Description

The Redirect() method of the Response object sends back an appropriate 302 HTTP header directive instructing the browser to request and load the *url* passed.

Example

Listing 9.183 shows how you can redirect the browser to a new page if a user accesses your root directory page (for example, www.purejavascript.com).

Listing 9.183 *Using the Redirect() Method*

```
<script runat="server" type="text/jscript" language="JScript">
Response.Redirect("/welcome.asp");
</script>
```

Response.Status

ASP1+

Syntax

Response.Status = *string*

Description

The Status property of the Response object specifies the HTTP status of the response.

Example

Listing 9.184 shows how you can return an error to the user if she has made an illegal request.

Listing 9.184 *Using the Status Property*

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Status = "404 Document Not Found";  
</script>
```

Response.Write()

ASP1+

Syntax

Response.Write(*string*)

Description

The Write() method of the Response object is used to write an ASCII *string* to the output.

Example

Listing 9.185 shows how you can write "Hello World" to output.

Listing 9.185 *Using the Write() Method*

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write("Hello World");  
</script>
```

Resultset

ES3+

Syntax

Core object created by the Stproc.resultSet().

Description

The Resultset object is one of the core server-side JavaScript objects. This object is created by the resultSet() method of the Stproc object. For stored procedures run against DB2, Oracle, Sybase, and ODBC databases, the stored procedure object has one Resultset object for each SQL SELECT statement executed by the procedure. An Informix stored procedure, on the other hand, has one Resultset object.

Each instance of this object has a property for each column in the resultset. For Oracle, Sybase, and ODBC stored procedures, these properties can be referred to by the column name. Informix and DB2 stored procedures do not have named columns, so you must use a zero-based numeric index to refer to the column.

You should always call the `close()` method of the `ResultSet` object after you have finished. The JavaScript runtime will attempt to close the object when the associate `DbPool` or database object goes out of scope.

NOTE

After creating a `resultset`, you must perform all operations on the set before calling the `commitTransaction()`, `returnValue()`, `outParameters()`, `Connection.cursor()`, or `Connection.SQLTable()` method.

This object has only one property—`prototype`. This property can be used to create new methods and properties for the `ResultSet` object. Table 9.29 contains the methods accessible from this object.

Table 9.29 *Methods of the `ResultSet` Object*

Method	Description
<code>close()</code>	Closes the <code>ResultSet</code> object and frees any memory used
<code>columnName()</code>	Takes an indexed numbered location and returns the column name of the column in that location
<code>columns()</code>	Returns the number of columns in the <code>ResultSet</code>
<code>next()</code>	Moves from the current row in the <code>ResultSet</code> object to the next row
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property

Example

Listing 9.186 creates a connection to a database. After the connection has been verified, the `storedProc()` method is used to invoke the fictitious `sp_employees` stored procedure. The resultsets for this stored procedure are then held in an instance of the `ResultSet` object.

Listing 9.186 *The `ResultSet` Object*

```
<server>
```

```
// Open a connection.
```

```
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
```

```
if(myConn.connected()){
```

```
    myConn.beginTransaction();
```

```
    // Run the stored procedure.
```

```
    var myStproc = myConn.storedProc("sp_employees");
```

```
    var myResultSet = myStproc.resultSet();
```

Listing 9.186 Continued

```
// Commit the transaction.
myConn.commitTransaction();

// Close the result set.
myResultSet.close();

// Release the connection.
myConn.release();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

ResultSet.close()

ES3+

Syntax

```
resultSet.close()
```

Description

The `close()` method of the `ResultSet` object closes the resultset and frees all memory used to store its information. If successful, the method returns `0`; otherwise, it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or database objects.

Example

Listing 9.187 creates a resultset and then closes it.

Listing 9.187 Closing a Resultset with the `close()` Method

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

myConn.beginTransaction();

// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Store the resultset.
myResultSet = myStproc.resultSet();
```



```
// Commit the transaction.
myConn.commitTransaction();
```

```
// Close the resultset.
myResultSet.close();
```

```
</server>
```

ResultSet.columnName()

ES3+

Syntax

```
resultSet.columnName(num)
```

Description

The `columnName()` method of the `ResultSet` object takes the zero-based indexed number location, *num*, passed to the method and returns the name of the column in that location. Note that these names are not returned in any specific order unless you order them as such. Successive calls to the method, however, will return all the columns. See the example for more information.

Example

Listing 9.188 has two resultsets. One of the sets returns specific column names and the other returns all columns. See the comments in the code for the output.

Listing 9.188 *Using the `columnName()` Method to Return the Names of the Columns in a Table*

```
<server>
```

```
// Create a pool of connections.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

// Open a connection from the pool. Give error if connection could
// not be made.
var myConn = myPool.connection('Employees', 15);

if(myConn.connected()){

    // Start a new SQL transaction.
    myConn.beginTransaction();

    // Run the stored procedure.
    var myStproc = myConn.storedProc("sp_employees");
```

Listing 9.188 Continued

```
// Store the resultsets.
var myResultSet1 = myStproc.resultSet();
var myResultSet2 = myStproc.resultSet();

// Writes column #1, from the first resultset, to the page.
write(myResultSet1.columnName(0));

// Writes column #2, from the first resultset, to the page.
write(myResultSet1.columnName(1));

// Writes all column names stored in the second resultset to the page.
for(var i = 0; i <= myResultSet2.columns(); i++){
    write(myResultSet2.columnName(i));
}

// Close the resultsets.
myResultSet1.close();
myResultSet2.close();

// End SQL transaction.
myConn.commitTransaction();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

// Release the connection
myConn.release();

</server>
```

ResultSet.columns()

ES3+

Syntax

```
resultSet.columns()
```

Description

The `columns()` method of the `ResultSet` object returns the number of columns in the resultset on which it is invoked. If the SQL used to create the resultset specifies a set number of columns to return, that is the number returned by this method.

Example

Listing 9.189 shows how you can return all the column names of the columns in your resultset.

Listing 9.189 Using the `columns()` Method to Determine How Many Columns Are in the Resultset Before Writing Them to the Page

```

<server>

// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Store the resultset.
var myResultSet = myStproc.resultSet();

// Writes all column names stored in the resultset to the page.
for(var i = 0; i < myResultSet.columns(); i++){
    write(myResultSet.columnName(i));
}

// Close the resultset.
myResultSet.close();

</server>

```

ResultSet.next()

ES3+

Syntax

```
resultset.next()
```

Description

The `next()` method of the `ResultSet` object moves the pointer in the current row to the next row in the resultset. This method is used to iterate through each of the rows returned by the resultset. This method returns `true`, unless it is the last row of the resultset; at which time it returns `false`.

Example

Listing 9.190 creates an instance of the `ResultSet` object and iterates through its results. This is performed by using the `next()` method.

Listing 9.190 Using the `next()` Method to Iterate Through the Rows in a Resultset

```

<server>

// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Store the resultset.
var myResultSet = myStproc.resultSet();

```

Listing 9.190 Continued

```
// Iterate through each return of the resultSet.
while(myResultSet.next()){

    // Perform processing here.

}

// Close the resultSet.
myResultSet.close();

</server>
```

ResultSet.prototype

ES3+

Syntax

```
resultSet.prototype.method = name
resultSet.prototype.property = value
```

Description

The prototype property of the `ResultSet` object allows you to create new properties and methods of the object. If you are adding a method, set the instance equal to the *name* of the method you have defined.

Example

Listing 9.191 creates a new property and method of the `ResultSet` object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to the page.

Listing 9.191 Using the prototype *Property* to Create a New Property and Method

```
<server>

// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
    }else{
        return false;
    }
}
```

```
// Create a new property and method of the Resultset object.
ResultSet.prototype.type = null;
ResultSet.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultset.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");
var myResultSet = myStproc.resultSet();

// Using the prototype we defined, assign the type property.
myResultSet.type = "work";

// Check the type of the resultset to see if it is valid.
if(myResultSet.isWORK()){
    write(myResultSet + " has a valid type of " + myResultSet.type);
}else{
    write(myResultSet + " has an invalid type of " + myResultSet.type);
}

</server> >
```

ResultSet.unwatch()

ES3+

Syntax

```
resultset.unwatch(property)
```

Description

The `unwatch()` method of the `ResultSet` object is used to turn off the watch for a particular *property*.

Example

Listing 9.192 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.192 Example of the `unwatch()` Method

```
<server>

// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
```

Listing 9.192 Continued

```
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + " ) changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the Resultset object.
Resultset.prototype.type = null;
Resultset.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultset.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");
var myResultSet = myStproc.resultSet();

// Using the prototype we defined, assign the type property.
myResultSet.type = "work";

// watch property
myResultSet.watch("type", alertme);

// Check the type of the resultset to see if it is valid.
if(myResultSet.isWORK()){
    write(myResultSet + " has a valid type of " + myResultSet.type);
}else{
    write(myResultSet + " has an invalid type of " + myResultSet.type);
}

// change value
myResultSet.type = null;

// turn off watch
myResultSet.unwatch("type");

// change value again
myResultSet.type = "home";

</server>
```

ResultSet.watch()

ES3+

Syntax

```
resultset.watch(property, function)
```

Description

The `watch()` method of the `ResultSet` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.193 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.193 *Example of the watch() Method*

<server>

```
// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
    }else{
        return false;
    }
}

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a new property and method of the ResultSet object.
ResultSet.prototype.type = null;
ResultSet.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultSet.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");
var myResultSet = myStproc.resultSet();
```

Listing 9.193 Continued

```
// Using the prototype we defined, assign the type property.
myResultSet.type = "work";

// watch property
myResultSet.watch("type", alertme);

// Check the type of the resultset to see if it is valid.
if(myResultSet.isWORK()){
    write(myResultSet + " has a valid type of " + myResultSet.type);
}else{
    write(myResultSet + " has an invalid type of " + myResultSet.type);
}

// change value
myResultSet.type = null;

</server>
```

ScriptingContext

ASP1+

Syntax

Core ASP environment object

Description

The ScriptingContext object is a core, but obsolete, ASP environment object. It will return the Application, Request, Response, Server, or Session built-in objects. Rather than using this object, you should useObjectContext instead.

Example

Please see the examples under the ObjectContext object for methods of accomplishing the same thing as this obsolete object.

SendMail

ES3+

Syntax

```
new SendMail()
```

Description

The SendMail object is a core server-side object that is created using the new keyword. This object provides the properties and methods necessary to send e-mail with your JavaScript applications. Table 9.30 lists the properties and methods of this object.

Table 9.30 SendMail *Methods and Properties*

Type	Item	Description
Property	Bcc	Property that contains the e-mail addresses of those users you want to blind carbon copy (Bcc)
	Body	Property that contains the actual body of the message
	Cc	Property that contains the e-mail addresses of those users you want to carbon copy
	Errorsto	Property that contains the e-mail address to which to send error messages
	From	Property that contains the sender's e-mail address
	Organization	Property that contains the sender's organization
	Replyto	Property that contains the sender's reply to e-mail address
	Smtpserver	Property that specifies the IP address or hostname of the mail server to send the message
	Subject	Property that contains the subject of the e-mail
	To	Property that contains the e-mail address of the recipient
Method	errorCode()	Method that returns an integer error code that might be incurred when sending e-mail
	errorMessage()	Method that returns a string related to any error messages that might be incurred when sending e-mail
	send()	Method that sends the e-mail
	unwatch()	Method that is used to turn off the watch for a particular property
	watch()	Method that is used to turn on the watch for a particular property

Using the SendMail object is very straightforward. Simply set the same properties contained in the everyday e-mail you send and invoke the `send()` method. If an error is encountered, it can be analyzed by using the error methods supplied.

Example

Listing 9.194 shows the use of the SendMail object to create a page for users to send e-mail.

Listing 9.194 Example of Using the SendMail Object

```
<html>
<head>
  <title>Using the SendMail object</title>
</head>
<body>
<server>
```

Listing 9.194 Continued

```
// See if they have submitted or just need the form.
if(request.method == "POST"){

    // Create an instance of the SendMail object.
    var myMail = new SendMail();

    // Assign the properties their values.
    myMail.To = request.toAddress;
    myMail.From = request.fromAddress;
    myMail.Subject = request.subject;
    myMail.Body = request.body;
    myMail.Smtpserver = "mail.purejavascript.com";
    myMail.Errorsto = "errors@purejavascript.com"

    // Try to send the mail.
    if(!myMail.send()){

        // If there was an error, give the user the email address of who they
        // should contact about the error, as well as the error code and message.
        write("There was an error sending your message. Please send email to ");
        write(myMail.Errorsto + " with the following error message");
        write("Error " + myMail.errorCode() + ": " + myMail.errorMessage());
    }else{

        // If there was not an error, tell the user they were successful.
        write("Your message was sent successfully!");
    }
}else{

    // If this page was called and a form was not submitted, then write the
    // email form to the page for the user to use.

    write('<form name="myForm" method="post">');
    write('<table border="1"><tr><td>');
    write('<table border="0">');
    write('<tr align="left" valign="top">');
    write('<td><b>To:</b></td>');
    write('<td><input type="text" name="toAddress" size="30"></td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td><b>From:</b></td>');
    write('<td><input type="text" name="fromAddress" size="30"></td>');
    write('</tr>');
    write('<tr align="left" valign="top">');
    write('<td><b>Subject:</b></td>');
    write('<td><input type="text" name="subject" size="30"></td>');
    write('</tr>');
```

```

write('<tr align="left" valign="top">');
write('<td><b>Body:</b></td>');
write('<td><textarea name="body" cols="60" rows="10" wrap="soft">');
write('</textarea></td>');
write('</tr>');
write('<tr align="left" valign="top">');
write('<td colspan=2 align="right">');
write('<input type="submit" value="Send Mail"></td>');
write('</tr>');
write('</table>');
write('</td></tr></table>');
write('</form>');
}
</server>
</body>
</html>

```

SendMail.Bcc

ES3+

Syntax

sendmail.Bcc

Description

The Bcc property of the SendMail object specifies the e-mail addresses of those recipients you want to blind carbon copy. If you want to specify more than one recipient, separate their e-mail addresses with commas.

Example

Listing 9.195 shows you how to set the Bcc property of an instance of the SendMail object.

Listing 9.195 *Setting the Bcc Property*

```

<server>

// Set the Bcc property.
myMail.Bcc = "publisher@purejavascript.com";

</server>

```

SendMail.Body

ES3+

Syntax

sendmail.Body

Description

The `Body` property of the `SendMail` object specifies the body of the e-mail you want to send.

Example

Listing 9.196 shows you how to set the `Body` property of an instance of the `SendMail` object.

Listing 9.196 Setting the Body Property

```
<server>

// Set the Body property.
myMail.Body = "Here is the text of the message";

</server>
```

SendMail.Cc

ES3+

Syntax

`sendmail.Cc`

Description

The `Cc` property of the `SendMail` object specifies the e-mail addresses of those recipients you want to carbon copy. If you want to specify more than one recipient, separate their e-mail addresses with commas.

Example

Listing 9.197 shows you how to set the `Cc` property of an instance of the `SendMail` object.

Listing 9.197 Setting the Cc Property

```
<server>

// Set the Cc property.
myMail.Cc = "techedit@purejavascript.com";

</server>
```

SendMail.constructor

ES2+

Syntax

`sendmail.constructor`

Description

The constructor property of the `SendMail` object specifies the function that creates the object.

Example

Listing 9.2198 shows an example of the constructor property.

Listing 9.198 *Example of the constructor Property*

```
<server>

// Create an instance of the SendMail object.
var myMail = new SendMail();

if(myMail.constructor == SendMail){
    write("Object created");
}

</server>
```

SendMail.errorCode()

ES3+

Syntax

```
sendmail.errorCode()
```

Description

The `errorCode()` method of the `SendMail` object returns one of six error codes after attempting to send an e-mail. These codes are described in Table 9.31.

Table 9.31 *Error Codes Generated from Sending E-mail with the `SendMail.send()` Method*

Code	Description
0	The e-mail was sent successfully.
1	The SMTP sending mail server was not specified.
2	The SMTP sending mail server was down or does not exist.
3	No recipient e-mail address was specified.
4	No sender's e-mail address was specified.
5	Connection problems and data not sent.

Example

Listing 9.199 tries to send an instance of the `SendMail` object. If the `send()` method fails, the error it encounters is written to the page.

Listing 9.199 Displaying the Error Code on a Failed E-mail Delivery Using the `errorCode()` Method.

```
<server>

// Try to send an instance of the mail.
if(!myMail.send()){

    // If there was an error, give the user the email address of who they
    // should contact about the error, as well as the error code and message.
    write("There was an error sending your message. Please send email to ");
    write(myMail.Errorsto + " with the following error message");
    write("Error " + myMail.errorCode() + ": " + myMail.errorMessage());

}else{

    // If there was not an error, tell the user they were successful.
    write("Your message was sent successfully!");
}

</server>
```

SendMail.errorMessage()

ES3+

Syntax

```
sendmail.errorMessage()
```

Description

The `errorMessage()` method of the `SendMail` object returns the string error message generated after attempting to send an e-mail.

Example

Listing 9.200 tries to send an instance of the `SendMail` object. If the `send()` method fails, the error it encounters is written to the page.

Listing 9.200 Displaying the Error Message on a Failed E-mail Delivery Using the `errorMessage()` Method.

```
<server>

// Try to send an instance of the mail.
if(!myMail.send()){

    // If there was an error, give the user the email address of who they
    // should contact about the error, as well as the error code and message.
    write("There was an error sending your message. Please send email to ");
```

```
write(myMail.Errorsto + " with the following error message");
write("Error " + myMail.errorCode() + ": " + myMail.errorMessage());

}else{

    // If there was not an error, tell the user they were successful.
    write("Your message was sent successfully!");
}

</server>
```

SendMail.Errorsto

ES3+

Syntax

`sendmail.Errorsto`

Description

The `Errorsto` property of the `SendMail` object specifies the e-mail address of a recipient that should receive the error message. The default value of this error is the sender's address. If you want to specify more than one recipient, separate their e-mail addresses with commas.

Example

Listing 9.201 shows you how to set the `Errorsto` property of an instance of the `SendMail` object.

Listing 9.201 *Setting the `Errorsto` Property*

```
<server>

// Set the Errorsto property.
myMail.Errorsto = "authors@purejavascript.com";

</server>
```

SendMail.From

ES3+

Syntax

`sendmail.From`

Description

The `From` property of the `SendMail` object specifies the e-mail address of the sender of the message.

Example

Listing 9.202 shows you how to set the From property of an instance of the SendMail object.

Listing 9.202 *Setting the From Property*

```
<server>

// Set the From property using the email key in a form sent by the client.
myMail.From = request.email;

</server>
```

SendMail.Organization

ES3+

Syntax

`sendmail.Organization`

Description

The Organization property of the SendMail object specifies the organization of the sender.

Example

Listing 9.203 shows you how to set the Organization property of an instance of the SendMail object.

Listing 9.203 *Setting the Organization Property*

```
<server>

// Set the Organization property.
myMail.Organization = "TIPs Technical Publishing";

</server>
```

SendMail.prototype

ES3+

Syntax

```
sendmail.prototype.method = name
sendmail.prototype.property = value
```

Description

The prototype property of the SendMail object allows you to create new properties and methods of the object. If you are adding a method, set the instance equal to the *name* of the method you have defined.

Example

Listing 9.204 creates a new property and method of the `SendMail` object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to the page.

Listing 9.204 *Using the prototype **Property** to Create a New Property and Method*

```
<server>

// Define the method that we prototyped.
function verifyAttach(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the SendMail object.
SendMail.prototype.attachment = null;
SendMail.prototype.hasAttach = verifyAttach;

// Create a SendMail object.
var myMail = new SendMail();

// Using the prototype we defined, assign the type property.
myMail.type = false;

// Check to see if there is an attachment.
if(myMail.hasAttach()){
    write(myMail + " has a valid type of " + myMail.type);
}else{
    write(myMail + " has an invalid type of " + myMail.type);
}

</server>
```

SendMail.Replyto

ES3+

Syntax

`sendmail.ReplyTo`

Description

The `ReplyTo` property of the `SendMail` object specifies the e-mail addresses to which the sender wants any replies to be routed.

Example

Listing 9.205 shows you how to set the `ReplyTo` property of an instance of the `SendMail` object.

Listing 9.205 *Setting the ReplyTo Property*

```
<server>

// Set the ReplyTo property using the email key in a form sent by the client.
myMail.ReplyTo = request.replyto;

</server>
```

SendMail.send()

ES3+

Syntax

```
sendmail.send()
```

Description

The `send()` method of the `SendMail` object attempts to send the e-mail from the value specified in the `From` property to those specified in the `To`, `Cc`, and `Bcc` properties through the e-mail server specified in the `Smtptserver` property. This method returns `true` if successful and `false` otherwise.

Example

Listing 9.206 tries to send a message. If an error occurs when trying to send the message, an error message is written to the user's page.

Listing 9.206 *Sending a Message with the send() Method*

```
<server>

// Try to send an instance of the mail.
if(!myMail.send()){

    // If there was an error, give the user the email address of whom they
    // should contact about the error, as well as the error code and message.
    write("There was an error sending your message. Please send email to ");
    write(myMail.Errorsto + " with the following error message");
    write("Error " + myMail.errorCode() + ": " + myMail.errorMessage());
}
```

```
}else{

    // If there was not an error, tell the user they were successful.
    write("Your message was sent successfully!");
}

</server>
```

SendMail.Smtpserver

ES3+

Syntax

`sendmail.Smtpserver`

Description

The Smtpserver property of the SendMail object specifies the SMTP sending email IP address or server name. This value defaults to the value set in the Administration Server settings for the instance of the Enterprise Server under which your application is running.

Example

Listing 9.207 shows you how to set the Smtpserver property of an instance of the SendMail object.

Listing 9.207 Setting the Smtpserver Property

```
<server>

// Set the Smtpserver property using the email key in a form
// sent by the client.
myMail.Smtpserver = request.smtp;

</server>
```

SendMail.Subject

ES3+

Syntax

`sendmail.Subject`

Description

The Subject property of the SendMail object specifies the subject of the e-mail you want to send.

Example

Listing 9.208 shows you how to set the Subject property of an instance of the `SendMail` object.

Listing 9.208 *Setting the Subject Property*

```
<server>

// Set the Subject property.
myMail.Subject = "I really liked your book!";

</server>
```

SendMail.To

ES3+

Syntax

`sendmail.To`

Description

The `To` property of the `SendMail` object specifies the e-mail addresses of the recipients of the message. If you want to specify more than one recipient, separate their e-mail addresses with commas.

Example

Listing 9.209 shows you how to set the `To` property of an instance of the `SendMail` object.

Listing 9.209 *Setting the To Property*

```
<server>

// Set the To property.
myMail.To = "authors@purejavascript.com";

</server>
```

SendMail.unwatch()

ES3+

Syntax

`sendmail.unwatch(property)`

Description

The `unwatch()` method of the `SendMail` object is used to turn off the watch for a particular *property*.

Example

Listing 9.210 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.210 Example of the `unwatch()` Method

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a SendMail object.
var myMail = new SendMail();

myMail.To = "allen@purejavascript.com";

// watch property
myMail.watch("To", alertme);

// change value
myMail.To = "jason@purejavascript.com";

// turn off watch
myMail.unwatch("To");

// change value again
myMail.To = "chuck@purejavascript.com";

</server>
```

SendMail.watch()

ES3+

Syntax

```
sendmail.watch(property, function)
```

Description

The `watch()` method of the `SendMail` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.211 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.211 Example of the watch() Method

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID ( " + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// Create a SendMail object.
var myMail = new SendMail();

myMail.To = "allen@purejavascript.com";

// watch property
myMail.watch("To", alertme);

// change value
myMail.To = "jason@purejavascript.com";

</server>
```

Server
ASP1+

Syntax

Core ASP environment object

Description

The Server object is a core ASP environment object. This object provides access to the server's utility functions. Table 9.32 contains the list of a property and methods of this object.

Table 9.32 A Property and Methods of the Server Object

Type	Item	Description
Methods	CreateObject()	Creates an instance of a server component.
	Execute()	Executes an .asp file.
	GetLastError()	Returns an ASPError object with the error condition.
	HTMLEncode()	Applies HTML encoding to the specified string.
	MapPath()	Maps either an absolute path on the server or the path relative to the current page, into a physical path.

Type	Item	Description
Property	<code>Transfer()</code>	Sends all the current state information to another .asp file for processing.
	<code>URLEncode()</code>	Applies URL encoding rules and escape characters to the string.
	<code>ScriptTimeout</code>	Amount of time a script can run before it times out.

Example

Listing 9.212 shows how you can use the `ScriptTimeout` property of the `Server` object to specify the number of seconds before a script stops processing.

Listing 9.212 Using the *Server* Object

```
<script runat="server" type="text/jscript" language="JScript">
Server.ScriptTimeout = 60;
</script>
```

Server.CreateObject()

ASP1+

Syntax

`Server.CreateObject(objID)`

Description

The `CreateObject()` method of the `Server` object creates an instance of the server component referenced by *objID*. This *objID* is usually in the following format:

Vendor.Component.Version

Example

Listing 9.213 creates an instance of a fictitious *Marketing* component from the *MyCo* company.

Listing 9.213 Using the *CreateObject()* Method

```
<script runat="server" type="text/jscript" language="JScript">
Server.CreateObject(MyCo.Marketing);
</script>
```

Server.Execute()

ASP3+

Syntax

`Server.Execute(path)`

Description

The `Execute()` method of the `Server` object executes the ASP file located at *path*, processing it as if it were part of the current page.

Example

Listing 9.214 executes the *gettime.asp* page while being processed.

Listing 9.214 Using the `Execute()` Method

```
<script runat="server" type="text/jscript" language="JScript">  
Server.Execute("/tools/gettime.asp");  
</script>
```

Server.GetLastError()

ASP3+

Syntax

`Server.GetLastError()`

Description

The `GetLastError()` method of the `Server` object creates an instance of the `ASPErrors` object that contains a description of the last error that occurred.

Example

Listing 9.215 writes out the last error that has occurred on a page.

Listing 9.215 Using the `GetLastError()` Method

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(Server.GetLastError().Description);  
</script>
```

Server.HtmlEncode()

ASP1+

Syntax

`Server.HtmlEncode(string)`

Description

The `HtmlEncode()` method of the `Server` object applies HTML encoding to a specified *string*.

Example

Listing 9.216 creates a string that includes two HTML tags, and before it is written to the page, HTML encoding is applied.

Listing 9.216 Using the `HTMLEncode()` Method

```
<script runat="server" type="text/jscript" language="JScript">
var myString = new String();
myString = "View the source to see no <b> tag in this <p>";
Response.Write(Server.HTMLEncode(myString));
</script>
```

Server.MapPath()

ASP1+

Syntax

`Server.MapPath(path)`

Description

The `MapPath()` method of the `Server` object maps the relative or virtual *path* to a physical directory on the file system.

Example

Listing 9.217 will write out the file system path for the requested URL.

Listing 9.217 Using the `MapPath()` Method

```
<script runat="server" type="text/jscript" language="JScript">
Response.Write(Server.MapPath(Request.ServerVariables("PATH_INFO")));
</script>
```

Server.ScriptTimeout

ASP1+

Syntax

`Server.ScriptTimeout`

Description

The `ScriptTimeout` property of the `Server` object specifies the number of seconds that should elapse before the currently executing script times out.

Example

Listing 9.218 shows how you can use the `ScriptTimeout` property of the `Server` object to specify the number of seconds before a script stops processing.

Listing 9.218 Using the `ScriptTimeout` Property

```
<script runat="server" type="text/jscript" language="JScript">
Server.ScriptTimeout = 60;
</script>
```

Server.Transfer()

ASP3+

Syntax

`Server.Transfer(path)`

Description

The `Transfer()` method of the `Server` object sends all the processed information available for the current page to the ASP file located at *path*.

Example

Listing 9.219 shows how the current script passes its data on to another script.

Listing 9.219 Using the `Transfer()` Method

```
<script runat="server" type="text/jscript" language="JScript">

// Define item to transfer.
var version = "1.0";

Server.Transfer("/scripts/logversion.asp");
</script>
```

Server.URLEncode()

ASP1+

Syntax

`Server.URLEncode(string)`

Description

The `URLEncode()` method of the `Server` object applies URL encoding to a specified *string*.

Example

Listing 9.220 creates a string that includes two HTML tags and some whitespace. Before it is written to the page, URL encoding is applied.

Listing 9.220 Using the `URLEncode()` Method

```
<script runat="server" type="text/jscript" language="JScript">
var myString = new String();
myString = "View the source to see no <b> tag in this <p>";
Response.Write(Server.URLEncode(myString));
</script>
```

server

ES2+

Syntax

Core object created when the instance of Enterprise Server is started.

Description

The server object is one of the core server-side objects. An instance of this object is created for each instance of the Enterprise server you have running. This object is destroyed when the server process is stopped.

This object should be used to store global data you want to share and manage across your applications. Because of this, you will need to lock and unlock your code if you are changing any properties you have defined in your applications. The server object has the properties and methods listed in Table 9.33. Note that the properties are all read-only.

Table 9.33 *Properties and Methods of the server Object*

Type	Item	Description
Method	<code>lock()</code>	Locks your code while you perform data manipulation that should only have a single thread connected to it.
	<code>unlock()</code>	Unlocks previously locked code.
	<code>unwatch()</code>	Turns off the watch for a particular property.
	<code>watch()</code>	Turns on the watch for a particular property.
Property	<code>host</code>	Specifies the server name, sub-domain, and domain name of the Web server.
	<code>hostname</code>	Contains the same information as concatenating the <code>host</code> property with the <code>port</code> property and separating them with a colon.
	<code>port</code>	Specifies the port number on which the server is running.
	<code>protocol</code>	Contains the protocol portion of the URL. This includes the information up to the first colon, as in <code>http:.</code>

Example

Listing 9.221 uses the `lock()` and `unlock()` methods of the server object to lock the working code while the property, `totalHits`, is modified.

Listing 9.221 Accessing Properties of the `server` Object

```
<server>

// Lock the code and increment a server counter.
server.lock();
server.totalHits += 1;

// Unlock the code.
server.unlock();

</server>
```

server.hostES2+

Syntax`server.host`**Description**

The host property of the server object contains the server name, any sub-domain, and domain name of the Web server.

Example

Listing 9.222 writes the host of the server to the user's page.

Listing 9.222 Accessing the `host` Property

```
<server>

// Write the host property to the user's page.
write(server.host);

</server>
```

server.hostnameES2+

Syntax`server.hostname`**Description**

The hostname property of the server object contains the server name, any sub-domain, domain name, and port of the Web server. This property is the same as concatenating the host property with the port property and separating them with a colon.

Example

Listing 9.223 writes the hostname of the server to the user's page.

Listing 9.223 *Accessing the* `hostname` *Property*

```
<server>

// Write the hostname property to the user's page.
write(server.hostname);

</server>
```

`server.lock()`

ES2+

Syntax

```
server.lock()
```

Description

The `lock()` method of the `server` object locks the code in which you are working until you perform an `unlock`. If the code is already locked, this method will wait until it can get the lock, a timeout occurs, or an error occurs.

Example

Listing 9.224 sets a `server` object with a counter property, `totalHits`. Because you only want to increment the counter with each new user request on each application, it must be locked.

Listing 9.224 *Using the* `lock()` *Method to Lock Your Code*

```
<server>

// Lock the code and increment a server counter
server.lock();
server.totalHits += 1;

// Unlock the application
server.unlock();

</server>
```

`server.port`

ES2+

Syntax

```
server.port
```

Description

The port property of the server object contains the port number to which the Web server is responding.

Example

Listing 9.225 writes the port of the server to the user's page.

Listing 9.225 Accessing the port Property

```
<server>

// Write the port property to the user's page.
write(server.port);

</server>
```

server.protocol

ES2+

Syntax

server.protocol

Description

The protocol property of the server object contains the protocol to which the server is responding. This includes the information up to the first colon, as in http:.

Example

Listing 9.226 writes the protocol of the server to the user's page.

Listing 9.226 Accessing the protocol Property

```
<server>

// Write the protocol property to the user's page.
write(server.protocol);

</server>
```

server.unlock()

ES2+

Syntax

server.unlock()

Description

The unlock() method of the server object unlocks the code that you have locked. This method returns true if the unlocking was successful and false otherwise.

Example

Listing 9.227 sets a server object with a counter property, *hitCount*. Because you only want to increment the counter with each new user request, it must be locked. When the counter has been incremented, the project is unlocked.

Listing 9.227 *Using the* `unlock()` *Method to Unlock Your Code*

```
<server>

// Lock the code and increment a server counter.
server.lock();
server.hitCount += 1;

// Unlock the code.
server.unlock();

</server>
```

server.unwatch()

ES3+

Syntax

```
server.unwatch(property)
```

Description

The `unwatch()` method of the server object is used to turn off the watch for a particular *property*.

Example

Listing 9.228 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.228 *Example of the* `unwatch()` *Method*

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// watch property incase it changes
server.watch("hostname", alertme);

// turn off watch
server.unwatch("hostname");

</server>
```

server.watch()

ES3+

Syntax

```
server.watch(property, function)
```

Description

The `watch()` method of the `server` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.229 shows how the `watch()` method is used to start watching the user-defined property *p*.

Listing 9.229 *Example of watch() method*

```
<server>

// function that is called if property's value changes
function alertme(id, oldValue, newValue){
    write("ID (" + id + ") changed from " + oldValue + " to " + newValue);
    return newValue;
}

// watch property in case it changes
server.watch("hostname", alertme);

</server>
```

Session

ASP1+

Syntax

Core ASP environment object

Description

The `Session` object is a core ASP environment object. This object is used to store information about a particular user in a session. This information will last for the user's entire session, not just the current page she is on. The server automatically creates this object when a user without a session makes a request. Table 9.34 contains the list of collections, events, methods, and properties available from this object.

Table 9.34 *Contains the Collections, Events, Methods, and Properties Available from the Session Object*

Type	Item	Description
Collection	Contents	Contains items added to the session with script commands.
	StaticObjects	Contains objects created with the <object> tag and given session scope.
Event	Session_OnEnd	Declared in the Global.asa file.
	Session_OnStart	Declared in the Global.asa file.
Method	Abandon()	Destroys a Session object and releases its resources.
	Contents.Remove()	Deletes an item from the Contents collection.
	Contents.RemoveAll()	Deletes all items from the Contents collection.
Property	CodePage	Sets the code page that will be used for symbol mapping.
	LCID	Identifies Locale.
	SessionID	Returns the session identification for this user.
	Timeout	Specifies the timeout period for the session state.

Example

Listing 9.230 shows how you can use the Timeout property of the Session object to specify the number of minutes before a script session ends.

Listing 9.230 Using the Session Object

```
<script runat="server" type="text/jscript" language="JScript">
Session.Timeout = 15;
</script>
```

Session.Abandon()

ASP1+

Syntax

```
Session.Abandon()
```

Description

The Abandon() method of the Session object destroys all items that have been added to a given Session instance, and frees all resources they were consuming. If this method is not called explicitly, the server invokes it when a session has ended.

 NOTE

When this method is called, the resources and items are not cleared until the page finishes processing. So, it is possible to use items that have been *marked* for abandonment on the current page, but not on subsequent pages.

Example

Listing 9.231 creates an item for the session, destroys all items in the session, and then creates a new item. After this has completed, the script attempts to write both items to the page. Because these are both on the same page as the `Abandon()` method call, both will be written. However, the *version* item will not be accessible on subsequent pages in the same session.

Listing 9.231 Using the `Abandon()` Method

```
<script runat="server" type="text/jscript" language="JScript">

// Create a "version" item that will only last for this page.
Session("version") = "1.0";

// Destroy session.
Session.Abandon();

// Create a new item that will last until another Abandon()
// is called, or session ends.
Session("name") = "Allen";

// Try writing both to the page.
Response.Write(Session("version"));
Response.Write(Session("name"));
</script>
```

Session.CodePageASP1+

Syntax

```
Session.CodePage(codepage)
```

Description

The `CodePage` property of the `Session` object determines the code page that will be used for displaying dynamic content. Code page is a numeric value for the character set that is used to identify different languages and locales.

Example

Listing 9.232 sets the `CodePage` property to represent American English ANSI content.

Listing 9.232 Using the CodePage Property

```
<script runat="server" type="text/jscript" language="JScript">
Session.CodePage(1252);
</script>
```

Session.Contents

ASP1+

Syntax

```
Session.Contents(name)
```

Description

The Contents collection of the Session object contains a collection of items that have been created for the session instance without the use of the <object> element. You can access specific items by their *name*. This collection also has Remove() and RemoveAll() methods.

Example

Listing 9.233 shows how you can add items to a Session object instance, access specific items through the Contents collection, and then clear all the items using the RemoveAll() method.

Listing 9.233 Using the Contents Collection

```
<script runat="server" type="text/jscript" language="JScript">

// Add some items to the session.
Session("version") = "1.0";
Session("author") = "R. Allen Wyke";

// Access the "author" item.
Response.Write(Session.Contents("author"));

// Now remove all the items in the Session object.
Session.Contents.RemoveAll();
</script>
```

Session.Contents.Remove()

ASP1+

Syntax

```
Session.Contents.Remove(name)
Session.Contents.Remove(num)
```

Description

The `Remove()` method of the `Contents` collection of the `Session` object can be used to remove the *name* item from the collection. Additionally, you can specify the *num* index location and remove the items using this method as well.

Example

Listing 9.234 shows how you can add items to a `Session` object instance, access specific items through the `Contents` collection, and then clear the item using the `Remove()` method.

Listing 9.234 *Using the `Remove()` Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Add some items to the session.
Session("version") = "1.0";
Session("author") = "R. Allen Wyke";

// Access the "author" item.
Response.Write(Session.Contents("author"));

// Now remove the "author" item from the Session object.
Session.Contents.Remove("author");
</script>
```

Session.Contents.RemoveAll()

ASP1+

Syntax

```
Session.Contents.RemoveAll()
```

Description

The `RemoveAll()` method of the `Contents` collection of the `Session` object removes all items that have been added to the session.

Example

Listing 9.235 shows how you can add items to a `Session` object instance, access specific items through the `Contents` collection, and then clear all the items using the `RemoveAll()` method.

Listing 9.235 *Using the `RemoveAll()` Method*

```
<script runat="server" type="text/jscript" language="JScript">

// Add some items to the session.
Session("version") = "1.0";
Session("author") = "R. Allen Wyke";
```

```
// Access the "author" item.  
Response.Write(Session.Contents("author"));  
  
// Now remove all the items in the Session object.  
Session.Contents.RemoveAll();  
</script>
```

Session.LCID

ASP1+

Syntax

Session.LCID

Description

The LCID property of the Session object sets the locale identifier to display dynamic content. This explicit setting will override any @LCID directives you might have set in your file.

Example

Listing 9.236 shows how you can set the locale identifier to British English.

Listing 9.236 Using the LCID Property

```
<script runat="server" type="text/jscript" language="JScript">  
Session.LCID = 2057;  
</script>
```

Session.SessionID

ASP1+

Syntax

Session.SessionID

Description

The SessionID property of the Session object contains the unique identifier generated by the server for the current Session object instance.

Example

Listing 9.237 writes out the value of the current SessionID property.

Listing 9.237 Using the SessionID Property

```
<script runat="server" type="text/jscript" language="JScript">  
Response.Write(Session.SessionID);  
</script>
```

Session.Session_OnEnd

ASP1+

Syntax

```
function Session_OnEnd(){  
    code  
}
```

Description

The Session_OnEnd event function of the Session object is called when the Session_OnEnd event is fired. This occurs when a session ends or times out.

Example

Listing 9.238 shows how you can append a string to the end of the log file of the last request of a session.

Listing 9.238 Using the Session_OnEnd Event

```
<script runat="server" type="text/jscript" language="JScript">  
function Session_OnEnd(){  
    Response.AppendToLog("Your session has ended");  
}  
</script>
```

Session.Session_OnStart

ASP1+

Syntax

```
function Session_OnStart(){  
    code  
}
```

Description

The Session_OnStart event function of the Session object is called when the Session_OnStart event is fired. This occurs when the session starts.

Example

Listing 9.239 shows how you can append a string to the end of the log file on the first request of a session.

Listing 9.239 Using the Session_OnStart Event

```
<script runat="server" type="text/jscript" language="JScript">  
function Session_OnStart(){  
    Response.AppendToLog("Your application has started");  
}  
</script>
```

Session.StaticObjects

ASP1+

Syntax

`Session.StaticObjects(name)`

Description

The StaticObjects collection of the Session object stores all the objects created with <object> elements within the scope of a given session. These are referenced by the *name* given to each instance of the elements.

Example

Listing 9.240 shows how you can grab the instance *myControl* from your current ASP page.

Listing 9.240 Using the StaticObjects Collection

```
<script runat="server" type="text/jscript" language="JScript">
Session.StaticObjects("myControl");
</script>
```

Session.Timeout

ASP1+

Syntax

`Session.Timeout`

Description

The Timeout property of the Session object specifies the number of minutes that should elapse before the current session times out.

Example

Listing 9.241 shows how you can use the Timeout property of the Session object to specify the number of minutes before a session ends.

Listing 9.241 Using the Timeout Property

```
<script runat="server" type="text/jscript" language="JScript">
Session.Timeout = 15;
</script>
```

ssjs_generateClientID()

ES3+

Syntax

`ssjs_generateClientID()`

Description

The `ssjs_generateClientID()` function is a top-level function that is not associated with any core server-side object. This function returns a unique identifier from the run-time engine. This allows you to track a user's session across several pages when using client-side maintenance. You might want to store the identifier as a property of the client object, but be careful to keep it from being accessed by other clients.

Example

Listing 9.242 shows how you can assign a property of the client object an identifier to track a user.

Listing 9.242 Using the `ssjs_generateClientID()` Function

```
<server>

// Store the identifier in a client property.
client.sessionID = ssjs_generateClientID();

</server>
```

`ssjs_getCGIVariable()`

ES3+

Syntax

`_ssjs_getCGIVariable(envVariable)`

Description

The `ssjs_getCGIVariable()` function is a top-level function that is not associated with any core server-side object. This function allows you to retrieve environment variables that are available to your CGI processes. When a variable cannot be found, the function returns `null`. Table 9.35 contains the default list of variables you can access.

Table 9.35 Environment Variables Accessible by the `ssjs_getCGIVariable()` Function

Variable	Description
<code>AUTH_TYPE</code>	The authorization type if the request is protected by a type of authorization. This returns values such as <code>basic</code> .
<code>HTTPS</code>	This shows if security is active on the server. This returns values of <code>ON</code> or <code>OFF</code> .
<code>HTTPS_KEYSIZE</code>	The number of bits in the key used to encrypt the session.
<code>HTTPS_SECRETKEYSIZE</code>	The number of bits used to generate the server's private key.
<code>PATH_INFO</code>	Path information to the file you want to retrieve. This would be something similar to <code>/sports/baseball.html</code> .

Variable	Description
PATH_TRANSLATED	The actual path to the file you want to retrieve. This would be something similar to <i>/netscape/suitespot/docs/sports/baseball.html</i> .
QUERY_STRING	Any information after a ? character in the URL.
REMOTE_ADDR	The IP address of the host submitting the request.
REMOTE_HOST	The hostname address of the host submitting the request. DNS must be turned on for this feature.
REMOTE_USER	The name of the local HTTP user of the Web browser, if access authorization is turned on for this URL.
REQUEST_METHOD	The type of request that is being made, such as GET or POST.
SCRIPT_NAME	The filename you are trying to retrieve, such as <i>index.html</i> .
SERVER_NAME	The hostname or IP under which the server is running.
SERVER_PORT	The port number to which the server is responding.
SERVER_PROTOCOL	The protocol version supported by the requesting client, such as HTTP/1.0.
SERVER_URL	The URL a user would have to enter to access the server, such as <i>http://www.purejavascript.com:6969</i> .

Example

Listing 9.243 returns the SERVER_URL variable to a property of the server object.

Listing 9.243 Retrieving an Environment Variable

```
<server>

// Store the identifier in a server property.
server.serverURL = ssjs_getCGIVariable(SERVER_URL);

</server>
```

ssjs_getClientID()

ES3+

Syntax

```
ssjs_getClientID()
```

Description

The `ssjs_getClientID()` function is a top-level function that is not associated with any core server-side object. This function returns a unique identifier from the runtime engine. This allows you to track a user's session across several pages when using server-side maintenance. When using this function, there is no need to store the identifier as a property of the `client` object.

Example

Listing 9.244 writes the identifier generated by the `ssjs_getClientID()` function to the page.

Listing 9.244 *Seeing the `ssjs_getClientID` Function*

```
<server>

// Store the identifier in a variable.
var myIdentifier = ssjs_getClientID();

// Write the ID to the page.
write(myIdentifier);

</server>
```

Stproc

ES3+

Syntax

Core object created by the `storedProc()` method of the database or `Connection` objects.

Description

The `Stproc` object is a core server-side object that is created by the `storedProc()` method of the database or `Connection` objects. Because of the nature of connections to a database, you should call the `close()` method of this object when you have completed your processing. This will free any and all memory used by the stored procedure. Otherwise, the object will be destroyed when the database or `Connection` objects go out of scope.

The object itself has only one property—prototype. This property can be used to add methods and properties to the object as needed. Table 9.36 lists the methods of the `Stproc` object.

Table 9.36 *Methods of the `Stproc` Object*

Method	Description
<code>close()</code>	Closes a stored procedure instance
<code>outParamCount()</code>	Returns the number of output parameters from the stored procedure
<code>outParameters()</code>	Returns the value of the output parameter passed to the method
<code>resultSet()</code>	Creates a new <code>ResultSet</code> object
<code>returnValue()</code>	Returns the return value for the stored procedure
<code>unwatch()</code>	Turns off the watch for a particular property
<code>watch()</code>	Turns on the watch for a particular property

Example

Listing 9.245 creates an instance of the `Stproc` object, via a `Connection`. `storedProc()` method call, and then closes it.

Listing 9.245 *Creating an Instance of the `Stproc` Object*

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

myConn.beginTransaction();

// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Commit the transaction.
myConn.commitTransaction();

// Close the resultset.
myConn.close();

</server>
```

Stproc.close()

ES3+

Syntax

```
stproc.close()
```

Description

The `close()` method of the `Stproc` object closes the stored procedure and frees all memory used to store its information. If successful, the method returns `0`; otherwise, it returns an error code that can be obtained by using the `majorErrorCode()` and `majorErrorMessage()` methods of the `Connection` or `database` objects.

Example

Listing 9.246 creates a stored procedure and then closes it.

Listing 9.246 *Closing a Cursor with the `close()` Method*

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

myConn.beginTransaction();
```

Listing 9.246 Continued

```
// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Store the resultset.
var myResultSet = myStproc.resultSet();

// Commit the transaction.
myConn.commitTransaction();

// Close the resultset.
myResultSet.close();

</server>
```

Stproc.outParamCount()

ES3+

Syntax

```
stproc.outParamCount()
```

Description

The `outParamCount()` method of the `Stproc` object returns the number of output parameters from the stored procedure. Be sure to call this method before calling the `outParameters()` method to make sure that parameters have been returned.



NOTE

Informix stored procedures do not have output parameters, so this method always returns 0 when run against an Informix database.

Example

Listing 9.247 opens a connection to the database and runs a stored procedure. Based on the number of output parameters, the script writes the parameters to the user's screen.

Listing 9.247 Using the `outParamCount()` Method

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

myConn.beginTransaction();

// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");
```

```
// Write the output parameters of this stored procedure to the
// user's page.
for(var i = 0; i < myStproc.outParamCount(); i++){
    write(myStproc.outParameters(i) + '<br>');
}

// Close the resultset.
myStproc.close();

// Commit the transaction.
myConn.commitTransaction();

</server>
```

Stproc.outParameters()

ES3+

Syntax

stproc.outParameters(num)

Description

The `outParameters()` method of the `Stproc` object returns the output parameter from the stored procedure that is indexed at the *num* location. The return value of this method can be a string, number, double, or object. Be sure to call the `outParamCount()` method before calling this method to make sure that parameters have been returned.

NOTE

Informix stored procedures do not have output parameters, so there is no need to run this method.

Example

Listing 9.248 opens a connection to the database and runs a stored procedure. Based on the number of output parameters, the script writes the parameters to the user's screen.

Listing 9.248 Using the `outParameters()` Method

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

myConn.beginTransaction();
```

Listing 9.248 Continued

```
// Run the stored procedure.
var myStproc = myConn.storedProc("sp_employees");

// Write the output parameters of this stored procedure to the
// user's page.
for(var i = 0; i < myStproc.outParamCount(); i++){
    write(myStproc.outParameters(i) + '<br>');
}

// Close the resultset.
myStproc.close();

// Commit the transaction.
myConn.commitTransaction();

</server>
```

Stproc.prototype

ES2+

Syntax

```
stproc.prototype.method = name
stproc.prototype.property = value
```

Description

The prototype property of the Stproc object allows you to create new properties and methods of the object. If you are adding a method, set the instance equal to the *name* of the method you have defined.

Example

Listing 9.249 creates a new property and method of the Stproc object. An instance is created and the new property is set. The new method is then called to verify the property, and, if it is incorrect, an error message is written to the page.

Listing 9.249 Using the prototype Property to Create a New Property and Method

```
<server>

// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
    }
}
```

```

    }else{
        return false;
    }
}

// Create a new property and method of the Stproc object.
Stproc.prototype.type = null;
Stproc.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultset.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");

// Using the prototype we defined, assign the type property.
myStproc.type = "work";

// Check the type of the stored procedure to see if it is valid.
if(myStproc.isWORK()){
    write(myStproc + " has a valid type of " + myStproc.type);
}else{
    write(myStproc + " has an invalid type of " + myStproc.type);
}

</server>

```

Stproc.resultSet()

ES3+

Syntax

```
stproc.resultSet()
```

Description

The `resultSet()` method of the `Stproc` object creates a `Resultset` object for storing the results of running a stored procedure. For stored procedures run against DB2, Oracle, Sybase, and ODBC databases, the stored procedure object has one `Resultset` object for each SQL `SELECT` statement executed by the procedure. An Informix stored procedure, on the other hand, has one `Resultset` object.

Each instance of this object has a property for each column in the result set. For Oracle, Sybase, and ODBC stored procedures, these properties can be referred to by the column name. Informix and DB2 stored procedures do not have named columns, so you must use a zero-based numeric index to refer to the column.

You should always call the `close()` method of the `Stproc` object after you have finished. The JavaScript runtime will attempt to close the object when the associated `DbPool` or database object goes out of scope.

NOTE

After creating a resultset, you must perform all operations on the set before calling the `commitTransaction()`, `returnValue()`, `outParameters()`, `Connection.cursor()`, or `Connection.SQLTable()` method.

Example

Listing 9.250 creates a connection to a database. After the connection has been verified, the `storedProc()` method is used to invoke the fictitious `sp_employees` stored procedure. The resultsets for this stored procedure are then held in an instance of the `ResultSet` object that was created with the `resultSet()` method.

Listing 9.250 Calling the *resultSet()* Method

```
<server>

// Open a connection.
var myConn = database.connect("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");

if(myConn.connected()){

    myConn.beginTransaction();

    // Run the stored procedure.
    var myStproc = myConn.storedProc("sp_employees");

    myResultSet = myStproc.resultSet();

    // Commit the transaction.
    myConn.commitTransaction();

    // Close the resultset.
    myResultSet.close();

    // Release the connection.
    myConn.release();

// If the connection fails, write an error message.
}else{
    write('Error ('+myConn.majorErrorCode()+'): ' + myConn.majorErrorMessage());
}

</server>
```

Stproc.returnValue()

ES3+

Syntax

```
stproc.returnValue()
```

Description

The `returnValue()` method of the `Stproc` object returns the return value of the stored procedure. For DB2, Informix, and ODBC, this method always returns `null`. Oracle only returns `null` if the stored procedure did not return a value. Sybase always returns a value.

NOTE

Before you call this method, you must retrieve any `ResultSet` objects. When this method is called, no more data can be obtained from the current resultset and no more resultsets can be created.

Example

Listing 9.251 calls a stored procedure, creates a resultset, and obtains the result value. This value is then written to the user's page.

Listing 9.251 Accessing a Stored Procedure's Return Value Using the `returnValue()` Method

```
<server>

// Call the stored procedure.
var myStproc = database.storedProc("sp_employees");

// Generate a resultset.
var myResultSet = myStproc.resultSet();

// Get the return value. Note that you can no longer
// reference the myResultSet variable.
var myReturnValue = myStproc.returnValue();

// Write the resultset to the page.
write(myReturnValue);

</server>
```

Stproc.unwatch()

ES3+

Syntax

```
stproc.unwatch(property)
```

Description

The `unwatch()` method of the `Stproc` object is used to turn off the watch for a particular *property*.

Example

Listing 9.252 shows how the `unwatch()` method is used to stop watching the user-defined property *p*.

Listing 9.252 *Example of the* `unwatch()` *Method*

```
<server>
```

```
// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the Stproc object.
Stproc.prototype.type = null;
Stproc.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultset.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");

// Using the prototype we defined, assign the type property.
myStproc.type = "work";

// Check the type of the stored procedure to see if it is valid.
if(myStproc.isWORK()){
    write(myStproc + " has a valid type of " + myStproc.type);
}else{
    write(myStproc + " has an invalid type of " + myStproc.type);
}

// watch property
myStproc.watch("type", alertme);

// change value
myStproc.type = null;
```

```
// turn off watch
myStproc.unwatch("type");

// change value again
myStproc.type = "home";

</server>
```

Stproc.watch()

ES3+

Syntax

```
stproc.watch(property, function)
```

Description

The `watch()` method of the `stproc` object is used to turn on the watch for a particular property specified by *property*. Any time the specified *property* is changed after the `watch()` method has been called, the specified *function* is called.

Example

Listing 9.253 shows how the `watch()` method is used to start watching the user defined property *p*.

Listing 9.253 *Example of the watch() Method*

```
<server>

// Define the method that we prototyped.
function verifyWork(){

    // Check to see if the type property we added is set to a valid value.
    if(this.type == "work"){
        return true;
    }else{
        return false;
    }
}

// Create a new property and method of the Stproc object.
Stproc.prototype.type = null;
Stproc.prototype.isWORK = verifyWork;

// Create a pool of connections, a connection, a stored procedure,
// and a resultset.
var myPool = new DbPool("ORACLE", "mySID", "myApp", "appsPWD", "myTNS");
var myConn = myPool.connection('Employees', 15);
var myStproc = myConn.storedProc("sp_employees");
```

Listing 9.253 Continued

```
// Using the prototype we defined, assign the type property.
myStproc.type = "work";

// Check the type of the stored procedure to see if it is valid.
if(myStproc.isWORK()){
    write(myStproc + " has a valid type of " + myStproc.type);
}else{
    write(myStproc + " has an invalid type of " + myStproc.type);
}

// watch property
myStproc.watch("type", alertme);

// change value
myStproc.type = null;

</server>
```

write()

ES2+

Syntax`write(string)``write(num)``write(expression)`**Description**

The `write` function is a top-level function that is not associated with any object. This function writes information to the HTML page the script is generating to send back to the client. This function can take and write a string, numeric value, or an expression that returns an alphanumeric result. To write data to a file, see the entry for `File.write()` and `File.writeln()`.

TIP

The JavaScript runtime buffers all write data until 64KB have been collected. Then the buffer is flushed to the file. You can manually call the top-level `flush` function to improve performance of your pages, if they are waiting on database query results, by calling the function before you run the query.

Example

Listing 9.254 writes a string "Hello, World!" to the user's page.

Listing 9.254 *Using the write Function*

```
<server>

// Write a string to the page
write('Hello, World!');

</server>
```




CHAPTER 10

DOM Core

This chapter contains a reference to all of the DOM Level 2 Core language bindings for ECMAScript (that is, Netscape JavaScript and MS JScript). The DOM Level 2 Core API provides the programmer with tools to create valid and well-formed XML documents. In addition, functionality exists in the API to navigate XML documents as well as modify the underlying structure. For those familiar with the DOM Java language bindings, you will notice that the ECMAScript bindings bear a striking resemblance to the Java DOM API.

Attr

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

Attr represents an attribute in an Element. Attr inherits all methods and properties from Node. DOM considers Attrs not to be part of the document tree. Therefore, *parentNode*, *previousSibling*, and *nextSibling* are all *null*. Table 10.1 lists all the properties of the Attr object.

Table 10.1 *Arguments Associated With Attr Object*

Type	Item	Description
Property	name	Returns the name of this attribute.
	ownerElement	Returns the Element node that this attribute is attached to.
	specified	Returns True if this attribute was explicitly given a value in the original document.
	value	Returns the value of this attribute as a string.

Example

An Attr is created by using the CreateAttribute() method of the Document object.

Attr.Name

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

attrObj.name

Description

The name property of the Attr object returns the name of this attribute.

Example

Listing 10.1 iterates through a list of attributes and writes out each attribute's name.

Listing 10.1 *Writing an Attribute's Name Using the name Property of the Attr Object*

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var i = 0;
while(i < attrList.length) {
    var attrObj = attrList.item(i);
    writeAttrName(attrObj.name);
    i++;
}
// -->
</script>
</html>
```

Attr.specified

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`attrObj.specified`

Description

The read-only property `specified` of the `Attr` object returns true if this attribute's value was explicitly assigned in the original document.

Example

Listing 10.2 iterates through a list of attributes checking to see which ones were specified in the original XML document.

Listing 10.2 *Checking if Attribute Was Specified Using the `specified` Property of the `Attr` Object*

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var i = 0;
while(i < attrList.length) {
    var attrObj = attrList.item(i);
    if(attrObj.specified)
        handleSpecifiedAttr(attrObj);
    i++;
}
// -->
</script>
</html>
```

Attr.value

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`attrObj.Value`

Description

The `value` property of the `Attr` object returns the value for this attribute as a string. A `DOMException` object can be raised with value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.3 checks the value of a size attribute.

Listing 10.3 *Checking the Value of an Attribute Using the value Property of the Attr Object*

```
<html>
<script language="JScript">
<!--
var sizeValue = sizeAttrObj.value;
// -->
</script>
</html>
```

Attr.ownerElement

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`attrObj.ownerElement`

Description

The `ownerElement` property of the `Attr` object returns the `Element` node that this attribute is attached to.

Example

Listing 10.4 retrieves the owner element for an attribute and then removes the attribute from the element.

Listing 10.4 *Retrieving the Owner Element Using the ownerElement Property of the Attr Object*

```
<html>
<script language="JScript">
<!--
var elementObj = attrObj.ownerElement;
elementObj.removeAttribute(attrObj);
// -->
</script>
</html>
```

CDATASection

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

The CDATASection object inherits all methods and properties from the Text object and the CharacterData object. The CDATASection encapsulates XML CDATA sections that are used to escape blocks of text that shouldn't be interpreted as markup.

Example

A CDATASection object is instantiated using the CreateCDATASection() method of the Document object.

CharacterData

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

This object is inherited by other objects that need to access character data in DOM. CharacterData inherits all properties and methods from Node in addition to those listed in Table 10.2.

Table 10.2 Arguments Associated with CharacterData Object

Type	Item	Description
Property	length	This read-only property returns the length of the character data.
	data	This read-only property returns character data in the form of a string.
Method	substringData()	Returns a substring of the character data.
	appendData()	Appends a string to the end of the character data of the node.
	insertData()	Inserts a string at the specified offset.
	deleteData()	Removes a range of 16-bit units from the node.
	replaceData()	Replaces the characters starting at the specified offset with the specified string.

Example

CharacterData objects are never directly instantiated. They exist through subclasses such as Text and Comment.

CharacterData.data

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

characterDataObj.data

Description

The readonly property data of the CharacterData object returns the character data of this node.

Example

Listing 10.5 retrieves string data from a Text node and displays it.

Listing 10.5 Retrieving String Data from a Text Node Using the data Property of the CharacterData Object

```
<html>
<script language="JScript">
<!--
var textData = textObj.data;
write(textData);
// -->
</script>
</html>
```

CharacterData.length

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

characterDataObj.length

Description

The read-only property length of the CharacterData object returns the length of data in 16-bit units.

Example

Listing 10.6 queries a Text node for the length of its data field.

Listing 10.6 Checking Text Data Length Using the length Property of the CharacterData Object

```
<html>
<script language="JScript">
<!--
```

```
var dataSize = textObj.length;
if(dataSize > bufferSize)
    printError();
// -->
</script>
</html>
```

CharacterData.substringData()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`characterDataObj.substringData(offset, count)`

Description

The `substringData()` method of the `CharacterData` object returns a range of data from the node with the offset character at position *offset* and at length of *count*. A `DOMException` object can be raised with value `INDEX_SIZE_ERR` if the specified offset is negative or greater than the *Length*, or with value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.7 creates a substring from the given Text node's data.

Listing 10.7 Creating a Substring Using the `substringData()` Method of the `CharacterData` Object

```
<html>
<script language="JScript">
<!--
textObj.data = "Test Text Data";
var subStringData = textObj.subStringData(0, 4);
// -->
</script>
</html>
```

CharacterData.appendData()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`characterDataObj.appendData(arg)`

Description

The `appendData()` method of the `CharacterData` object appends the string *arg* to the end of *Data*. A `DOMException` object can be raised with value `NO_MODIFICATION_ALLOWED_ERR` if the node is read-only.

Example

Listing 10.8 adds a string to the end of a Text node's data property.

Listing 10.8 *Appending Data Using the `appendData()` Method of the `CharacterData` Object*

```
<html>
<script language="JScript">
<!--
textObj.data = "Test Text Data";
textObj.appendData("Some more data");
// -->
</script>
</html>
```

CharacterData.insertData()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`characterDataObj.insertData(offset, arg)`

Description

The `insertData()` method of the `CharacterData` object inserts a string *arg* at the specified position *offset*. A `DOMException` object can be raised with value `INDEX_SIZE_ERR` if the specified offset is negative or greater than Length or with the value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.9 inserts a string into a Text node's Data property.

Listing 10.9 *Inserting Data Using the `insertData()` Method of the `CharacterData` Object*

```
<html>
<script language="JScript">
<!--
textObj.data = "Test Text Data";
textObj.insertData(4, "Some more data");
// -->
</script>
</html>
```

CharacterData.deleteData()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`characterDataObj.deleteData(offset, count)`

Description

The `deleteData()` method of the `CharacterData` object removes a range of data from the node beginning at position *offset* and removing *count* characters. A `DOMException` object can be raised with the value `INDEX_SIZE_ERR` if the specified offset is negative or exceeds `Length` or with the value `NO_MODIFICATION_ALLOWED_ERR` raised if the node is read-only.

Example

Listing 10.10 deletes a string from a `Text` node's `Data` property.

Listing 10.10 *Deleting Data Using the `deleteData()` Method of the `CharacterData` Object*

```
<html>
<script language="JScript">
<!--
textObj.data = "Test Text Data";
textObj.deleteData(0, 5);
// -->
</script>
</html>
```

CharacterData.replaceData()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`characterDataObj.replaceData(offset, count, arg)`

Description

The `replaceData()` method of the `CharacterData` object replaces the characters of data contained in this node beginning at position *offset* and length *count* with the string *arg*.

Example

Listing 10.11 illustrates replacing one string for another in a `Text` node's `Data` property.

Listing 10.11 *Replacing Data Using the* `replaceData()` *Method of the* `CharacterData` **Object**

```
<html>
<script language="JScript">
<!--
textObj.data = "Test Text Data";
textObj.replaceData(0, 4, "foo ");
// -->
</script>
</html>
```

Comment

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

Comment inherits all methods and properties from the `CharacterData` object and represents the contents of an XML comment.

Example

A Comment is instantiated using the `createComment()` method of the Document object.

Document

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

A Document object is created using the `CreateDocument()` method of the `DOMImplementation` object. Table 10.3 lists all properties and methods of the Document object.

Table 10.3 *Arguments Associated with* Document **Object**

Type	Item	Description
Property	<code>docType</code>	Returns the document type associated with this XML document.
	<code>documentElement</code>	Returns the root document element for this XML document.

Type	Item	Description
Method	implementation	Returns the DOM implementation associated with this XML document.
	createAttribute()	Creates and returns a named attribute for this document.
	createAttributeNS()	Creates and returns an attribute.
	createCDATASection()	Returns a newly created CDATASection object.
	createComment()	Returns a newly created Comment object associated with this document.
	createDocumentFragment()	Returns a fragment of this XML document.
	createElement()	Returns a new element for this document.
	createElementNS()	Creates and returns an element.
	createEntityReference()	Creates and returns an XML Entity Reference.
	createProcessingInstruction()	Creates and returns a new processing instruction for this document.
	createTextNode()	Returns a newly created TextNode object associated with this document.
	getElementById()	Returns an element with the specified ID.
	getElementsByTagName()	Returns an element with the specified tag name.
	getElementsByTagNameNS()	Returns an element with a given tag name in a specified namespace.
	importNode()	Imports a specified node into this document.

Example

Listing 10.12 uses a DOMImplementation object to create a Document object.

Listing 10.12 Creating a Document

```

<html>
<script language="JavaScript">
<!--
var domImplObj = new DOMImplementation();
var documentObj = domImplObj.
    createDocument("http://foobar.org/schema", "foo:bar", null);
//-->
</script>
</html>

```

Document.createAttribute()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.createAttribute(name)`

Description

The `createAttribute()` method of the Document object creates and returns an Attr object with the given name. A DOMException object can be raised with value `INVALID_CHARACTER_ERR` if the specified name contains an illegal character.

Example

Listing 10.13 illustrates adding a new attribute to the root element of a previously created document object.

Listing 10.13 Adding an Attribute with the `createAttribute()` Method

```
<html>
<script language="JScript">
<!--
var attrObj = documentObj.createAttribute("size");
var rootElement = documentObj.rootElement;
attrObj.value = 34;
rootElement.setAttributeNode(attrObj);
// -->
</script>
</html>
```

Document.createAttributeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.createAttributeNS(namespaceURI, qualifiedName)`

Description

The `createAttributeNS()` method of the Document object creates and returns an Attr of type *qualifiedName* residing in the namespace *nameSpaceURI*. The Attr returned has its *nodeName* set to *qualifiedName*, its *nameSpaceURI* attribute set to *nameSpaceURI*, its *prefix* attribute set to a prefix as extracted from *qualifiedName* or *null* if there is no prefix, its *localName* attribute set to the local name as extracted from *qualifiedName*, its *name* attribute set to *qualifiedName*, and its *nodeValue* attribute set to empty string.

It should be noted that implementations aren't required to implement this method of Document. A `DOMException` object can be raised with the value `INVALID_CHARACTER_ERR` if *qualifiedName* contains an illegal character or a value of `NAME_SPACE_ERR` if *qualifiedName* is malformed, the *qualifiedName* has a prefix and *namespaceURI* is Null, or if *qualifiedName* has a prefix that is `xml` and the *namespaceURI* is other than `http://www.w3.org/XML/1998/namespace`.

Example

Listing 10.14 illustrates creating an attribute using namespace references.

Listing 10.14 Adding an Attribute

```
<html>
<script language="JScript">
<!--
var newAttribute = documentObj.createAttributeNS
    ("http://foo.com/namespace", "foo:quantity");
newAttribute.Value = 10;
myLineItemElement.SetAttributeNode(newAttribute);
// -->
</script>
</html>
```

Document.createCDATASection()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.createCDATASection(data)`

Description

The `createCDATASection()` method of the Document object creates and returns a new instance of `CDATASection` with its value set to *data*. A `DOMException` object with the value `NOT_SUPPORTED_ERR` can be raised if this document is an HTML document.

Example

Listing 10.15 illustrates how to create and add a CDATA section to a previously created document.

Listing 10.15 Adding a New CDATA Section

```
<html>
<script language="JScript">
<!--
var cdataSectionObj = documentObj.createCDATASection("foo");
// -->
</script>
</html>
```

Document.createComment()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
documentObj.createComment(data)
```

Description

The `createComment()` method of the Document object creates and returns a new Comment node using the DOMString *data*.

Example

Listing 10.16 uses a previously created Document object to create and add a new Comment to the current XML document.

Listing 10.16 Adding a New Comment to an XML Document

```
<html>
<script language="JScript">
<!--
var commentObj = documentObj.createComment
    ("Here is an example comment.");
// -->
</script>
</html>
```

Document.createDocumentFragment()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
documentObj.createDocumentFragment()
```

Description

The `createDocumentFragment()` method of the Document object creates an empty DocumentFragment object.

Example

Listing 10.17 uses a Document object to create a new DocumentFragment.

Listing 10.17 Creating a Document Fragment Using the createDocumentFragment() Method of Document

```
<html>
<script language="JScript">
<!--
```

```
var documentFragmentObj = documentObj.createDocumentFragment();
var nodeType = documentFragmentObj.NodeType;
if (nodeType != Node.DOCUMENT_FRAGMENT_NODE)
    printError();
// -->
</script>
</html>
```

Document.createElement()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.createElement(tagName)`

Description

The `createElement()` method of the Document object creates a new element of the type specified by *tagName*. This method returns a new instance of `Element` with its *nodeName* attribute set to *tagName* and its other attributes set to Null. A `DOMException` object can be thrown with value `INVALID_CHARACTER_ERR` if the specified *tagName* contains an illegal character.

Example

Listing 10.18 uses a Document object to create a new element.

Listing 10.18 *Checking Whether an Element Has Children Using the*
`hasChildren()` **Method of** `Element`

```
<html>
<script language="JScript">
<!--
var elementObj = documentObj.createElement("foo");
var hasChildren = elementObj.hasChildren();
// -->
</script>
</html>
```

Document.createElementNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.CreateElementNS(nameSpaceURI, qualifiedName)`

Description

The `createElementNS()` method of the Document object creates and returns an Element of type *qualifiedName* residing in the namespace *nameSpaceURI*. The Element returned has its *nodeName* set to *qualifiedName*, its *nameSpaceURI* attribute set to *nameSpaceURI*, its *prefix* attribute set to the *prefix* as extracted from *qualifiedName*, its *localName* attribute set to the *localName* as extracted from *qualifiedName*, and its *tagName* attribute set to *qualifiedName*. It should be noted that implementations aren't required to implement this method of Document. A `DOMException` object can be raised with the value `INVALID_CHARACTER_ERR` if *qualifiedName* contains an illegal character or a value of `NAMESPACE_ERR` if *qualifiedName* is malformed, the *qualifiedName* has a prefix and *nameSpaceURI* is null, or if *qualifiedName* has a prefix that is `xml` and the *nameSpaceURI* is other than `http://www.w3.org/XML/1998/namespace`.

Example

Listing 10.19 shows how to add an element to the current document.

Listing 10.19 Adding an Element

```
<html>
<script language="JScript">
<!--
var myLineItemElement = documentObj.createElementNS
    ("http://foo.com/namespace", "foo:lineItem");
// -->
</script>
</html>
```

Document.createEntityReference()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.createEntityReference(name)`

Description

The `createEntityReference()` method of the Document object creates and returns a new instance of an `EntityReference` object given the *name* of the entity to reference. If the named entity is already known, the child list of the `EntityReference` node is made the same as that of the corresponding entity. A `DOMException` object can be raised with the value `INVALID_CHARACTER_ERR` if the given name contains an illegal character or the value `NOT_SUPPORTED_ERR` if this document is an HTML document.

Example

Listing 10.20 illustrates using a previously created Document object to create an entity reference.

Listing 10.20 *Creating an EntityReference*

```
<html>
<script language="JScript">
<!--
var entityReferenceObj = documentObj.createEntityReference("myRef");
if(entityReferenceObj.hasAttributes())
    handleNoAttributes();
// -->
</script>
</html>
```

Document.createProcessingInstruction()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

documentObj.createProcessingInstruction(*target*, *data*)

Description

The createProcessingInstruction() method of the Document object creates an instance of a ProcessingInstruction node with DOMStrings *target* and *data* and returns it. A DOMException object with value INVALID_CHARACTER_ERR can be raised if the target contains an illegal character or with the value NOT_SUPPORTED_ERR if this document is an HTML document.

Example

Listing 10.21 illustrates adding a CDATA section to a previously created document.

Listing 10.21 *Adding a CDATA Section*

```
<html>
<script language="JScript">
<!--
var cdataSectionObj = documentObj.createCDATASection("foo CDATA Section");
// -->
</script>
</html>
```

Document.createTextNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

documentObj.createTextNode(*data*)

Description

The `createTextNode()` method of the Document object uses the string *data* to create a new text node for this Document. It returns a new instance of a Text object that has its *data* attribute set to *data*.

Example

Listing 10.22 uses a previously created Document object to create a new text node in the current document.

Listing 10.22 *Creating a New Text Node Using the* `createTextNode()` *Method of the* Document *Object*

```
<html>
<script language="JScript">
<!--
var textObj = documentObj.createTextNode("Some new text.");
var splitText = textObj.splitText(4);
// -->
</script>
</html>
```

Document.doctype

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.doctype`

Description

The `doctype` property of the Document object returns the document type declaration associated with this document. This attribute is read only and cannot be manipulated with methods inherited from Node such as `insertNode()` or `removeNode()`. The return type is `DocumentType`. For HTML documents, this property returns `Null`.

Example

Listing 10.23 shows how to check the document type declaration of the current document.

Listing 10.23 *Checking the* `doctype` *Property*

```
<html>
<script language="JScript">
<!--
var docTypeObj = documentObj.doctype;
if(docTypeObj == null)
    handleHTML();
else
    handleXML();
```

```
// -->
</script>
</html>
```

Document.documentElement

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.documentElement`

Description

The read-only property `documentElement` of the `Document` object allows direct access to the root element of this document. It's return value is type `Element`.

Example

Listing 10.24 illustrates using the `Document` object to retrieve the root element of the current document.

Listing 10.24 Retrieving the Root Document Element Using the `documentElement` Property of the `Document` Object

```
<html>
<script language="JScript">
<!--
var rootElementObj = documentObj.documentElement;
var hasFooAttribute = rootElementObj.hasAttribute("foo");
// -->
</script>
</html>
```

Document.getElementById()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.getElementById(elementId)`

Description

The `getElementById()` method of the `Document` object returns the `Element` whose ID is specified by *elementId* in this `Document`.

Example

Listing 10.25 shows how to retrieve an element using its element identifier.

Listing 10.25 Retrieving Element by Identifier

```
<html>
<script language="JScript">
<!--
var myElement = documentObj.getElementById("12932945");
// -->
</script>
</html>
```

Document.getElementsByTagName()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.getElementsByTagName(tagName)`

Description

The `getElementsByTagName()` method of the Document object returns a `NodeList` of all Elements with the specified *tagName*. The Elements returned in the `NodeList` occur in the order in which they were encountered in the Document. If *tagName* is set to `*`, the returned `NodeList` will contain a list of all Elements in this document.

Example

Listing 10.26 illustrates getting all elements in a previously created Document object and iterating through the list.

Listing 10.26 Getting Elements by Tag Name

```
<html>
<script language="JScript">
<!--
var elementListObj = documentObj.getElementsByTagName("item");
var i = 0;
while ( i < elementListObj.length) {
    processElementValue(elementListObj.item(i).nodeValue);
}
// -->
</script>
</html>
```

Document.getElementsByTagNameNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.getElementsByTagNameNS(namespaceURI, localName)`

Description

The `getElementsByTagNameNS()` method of the Document object returns a `NodeList` of all Elements in this Document with the given *localName* in the given name space as specified in *namespaceURI* in the order in which they are encountered in a preorder traversal of the Document.

Example

Listing 10.27 shows how to retrieve all elements in a document given a tag name.

Listing 10.27 *Getting Elements by Tag Name*

```
<html>
<script language="JScript">
<!--
var elementListObj = documentObj.getElementsByTagName
    ("http://foo.com/namespace", "foo:item");
var i = 0;
while ( i < elementListObj.length) {
    processElementValue(elementListObj.Item(i).NodeValue);
}
// -->
</script>
</html>
```

Document.implementation

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.implementation`

Description

The read-only property implementation of the Document object returns the `DOMImplementation` object associated with this document. This property is simply here for convenience.

Example

Listing 10.28 illustrates using a Document object to return the current `DOMImplementation`.

Listing 10.28 *Using the implementation Property*

```
<html>
<script language="JScript">
<!--
var domImplObj = documentObj.implementation;
var hasHTMLEvents = domImplObj.hasFeature("HTMLEvents", "2.0");
```

Listing 10.28 Continued

```
//-->
</script>
</html>
```

Document.importNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentObj.importNode(importedNode, deep)`

Description

The `importNode()` method of the `Document` object imports a `Node` specified in the parameter *importNode* from another document into this document and returns it. The returned `Node` has its *parentNode* attribute set to *null*. The *importNode* is not removed from the original document; it is simply copied into this document. If *deep* is set to *true*, the subtree beneath *importNode* will also be copied from the source document recursively. A `DOMException` object can be raised with value `NOT_SUPPORTED_ERR` if this document is an HTML document.

Example

Listing 10.29 shows a previously created node being imported into the current document and then made a child of an element.

Listing 10.29 Importing a Node

```
<html>
<script language="JScript">
<!--
var importedNode = documentObj.importNode(newNode, true);
elementObj.append(importedNode);
// -->
</script>
</html>
```

DocumentFragment

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

A DocumentFragment object represents a portion of an XML document's tree. It differs from Document in that it is lightweight in nature. It is useful for rearranging document structure in a simple fashion. It can be seen as analogous to the clipboard in cut-and-paste scenarios. The DocumentFragment object inherits all methods and properties of its parent Node.

Example

Listing 10.30 illustrates how a Document object is used to create a new DocumentFragment.

Listing 10.30 *Creating a Document Fragment Using the createDocumentFragment() Method of the Document Object*

```
<html>
<script language="JScript">
<!--
var docFragmentObj = documentObj.createDocumentFragment();
// -->
</script>
</html>
```

DocumentType

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

The DocumentType object provides access to the list of entities that are defined in the XML document's DTD. DocumentType inherits all properties and methods from Node. Table 10.4 lists all properties of the DocumentType object.

Table 10.4 *Arguments Associated With DocumentType Object*

Type	Item	Description
Property	entities	List of entities both internal and external contained in this DTD.
	internalSubset	A string representation of the internal subset (implementation dependant).
	name	Name of this document type (DTD).
	notations	List of notations declared in this DTD.
	publicId	The public identifier of the external subset.
	systemId	The system identifier of the external subset.

Example

A `DocumentType` object is created using the `CreateDocumentType()` method of the `DOMImplementation` object. The `DocumentType` represents the actual DTD used for building XML documents. It inherits all properties and methods of `Node`.

DocumentType.entities

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentTypeObj.entities`

Description

The read-only property `entities` of the `DocumentType` object returns a `NamedNodeMap` containing external and internal entities declared in the DTD.

Example

Listing 10.31 iterates through a list of entities and prints out their notation names.

Listing 10.31 Iterating Through the entities Property of the DocumentType Object

```
<html>
<script language="JScript">
<!--
var entityList = docTypeObj.entities;
var i = 0;
while( i < entityList.length) {
    writeEntityName(entityList.item(i).notationName);
}
// -->
</script>
</html>
```

DocumentType.internalSubset

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentTypeObj.internalSubset`

Description

The read-only property `internalSubset` of the `DocumentType` object returns a string representation of the internal subset.

Example

Listing 10.32 retrieves the internal subset from a `DocumentType` object.

Listing 10.32 *Reading the `internalSubset` Property of the `DocumentType` Object*

```
<html>
<script language="JScript">
<!--
var internalSubset = docTypeObj.internatSubset;
// -->
</script>
</html>
```

DocumentType.name

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentTypeObj.name`

Description

The read-only property name of the `DocumentType` object returns the name of this DTD.

Example

Listing 10.33 illustrates retrieving the Name of a DTD using the name property of the `DocumentType` object.

Listing 10.33 *Name Property of `DocumentType`*

```
<html>
<script language="JScript">
<!--
var docTypeName = docTypeObj.name;
// -->
</script>
</html>
```

DocumentType.notations

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentTypeObj.notations`

Description

The read-only property notations of the `DocumentType` object returns a `NamedNodeMap` containing all notations declared in the DTD.

Example

Listing 10.34 iterates through a list of notations and writes out their public identifiers.

Listing 10.34 *Reading the notations **Property of the** `DocumentType` Object*

```
<html>
<script language="JScript">
<!--
var notationList = docTypeObj.notations;
var i = 0;
while( i < entityList.length) {
    writeEntityName(entityList.item(i).publicId);
}
// -->
</script>
</html>
```

DocumentType.publicId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`documentTypeObj.publicId`

Description

The read-only property `publicId` of the `DocumentType` object returns the public identifier of the external subset.

Example

Listing 10.35 retrieves the public identifier from a `DocumentType` object.

Listing 10.35 *Reading the `publicId` **Property of the** `DocumentType` Object*

```
<html>
<script language="JScript">
<!--
var publicId = docTypeObj.publicId;
// -->
</script>
</html>
```

DocumentType.systemId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

documentTypeObj.systemId

Description

The read-only property `systemId` of the `DocumentType` object returns the system identifier of the external subset.

Example

Listing 10.36 retrieves the system identifier from a `DocumentType` object.

Listing 10.36 *Reading the `systemId` Property of the `DocumentType` Object*

```
<html>
<script language="JScript">
<!--
var systemId = docTypeObj.systemId;
// -->
</script>
</html>
```

DOMException

JavaScript 1.5+, JScript 5.0+

Syntax

Core DOM object.

Description

The `DOMException` object encapsulates all exception situations that are seen throughout the Core DOM DOM Level 2 API. Table 10.5 lists all of the exception types as well as the property of the `DOMException` object.

Table 10.5 *Arguments Associated With `DOMException` Object*

Type	Item	Description
Constant	<code>INDEX_SIZE_ERR</code>	Constant whose numeric value is 1.
	<code>DOMSTRING_SIZE_ERR</code>	Constant whose numeric value is 2.
	<code>HIERARCHY_REQUEST_ERR</code>	Constant whose numeric value is 3.
	<code>WRONG_DOCUMENT_ERR</code>	Constant whose numeric value is 4.
	<code>INVALID_CHARACTER_ERR</code>	Constant whose numeric value is 5.
	<code>NO_DATA_ALLOWED_ERR</code>	Constant whose numeric value is 6.
	<code>NO_MODIFICATION_ALLOWED_ERR</code>	Constant whose numeric value is 7.
	<code>NOT_FOUND_ERR</code>	Constant whose numeric value is 8.

Table 10.5 Continued

Type	Item	Description
Property	NOT_SUPPORTED_ERR	Constant whose numeric value is 10.
	INUSE_ATTRIBUTE_ERR	Constant whose numeric value is 10.
	INVALID_STATE_ERR	Constant whose numeric value is 11.
	SYNTAX_ERR	Constant whose numeric value is 12.
	INVALID_MODIFICATION_ERR	Constant whose numeric value is 13.
	NAMESPACE_ERR	Constant whose numeric value is 14.
	INVALID_ACCESS_ERR	Constant whose numeric value is 15.
	code	Number representing one of the previously listed exceptions.

Example

Listing 10.37 creates a new DOMException object with code value *NAMESPACE_ERR* and throws it.

Listing 10.37 Creating a DOMException

```
<html>
<script language="JScript">
<!--
var domExceptionObj = new DOMException(DOMException.NAMESPACE_ERR);
throw domExceptionObj;
// -->
</script>
</html>
```

DOMException.code

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

domExceptionObj.code

Description

The code property of the DOMException object returns a number indicating which exception has occurred.

Example

Listing 10.38 illustrates catching a DOMException object and interrogating the code property to handle the appropriate error.

Listing 10.38 *Catching a DOMException*

```

<html>
<script language="JScript">
<!--
var domImplObj = new DOMImplementation();
try {
    var documentObj = domImplObj.createDocument
        ("http://foobar.org/schema", "foo:bar", null);
} catch(e if e == "DOMException") {
    if (e.code == DOMException.INVALID_CHARACTER_ERR)
        handleError(e)
}
// -->
</script>
</html>

```

DOMImplementation

JavaScript 1.5+, JScript 5.0+
 Nav6+, IE5+

Syntax

DOM Core object.

Description

DOMImplementation is used to create XML documents and document types in a non-implementation specific manner. The object provides several convenient methods for performing tasks independent of the DOM implementation used. Table 10.6 lists all methods of the DOMImplementation object.

Table 10.6 *Arguments Associated with the DOMImplementation Object*

Type	Item	Description
Method	<code>createDocument()</code>	Creates a new XML document using the namespace URI, the qualified name of the new document, and the document type.
	<code>createDocumentType()</code>	Creates an empty document type using the new qualified name, the public ID for the new type, and the system ID for the new type.
	<code>hasFeature()</code>	Checks whether a specific feature is implemented in this DOM implementation.

Example

Listing 10.39 creates a DOMImplementation object and creates a Document.

Listing 10.39 Use of DOMImplementation

```
<html>
<script language="JScript">
<!--
var domImplObj = new DOMImplementation();
var documentObj = domImplObj.createDocument
    ("http://foobar.org/schema", "foo:bar", null);
// -->
</script>
</html>
```

DOMImplementation.createDocument()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOMImplementation.CreateDocument(*namespaceURI*, *qualifiedName*, *doctype*)

Description

The `createDocument()` method of the `DOMImplementation` object takes the two `DOMString` arguments *namespaceURI* and *qualifiedName* along with the `DocumentType` argument *doctype* to create and return a new XML Document object of the specified type. This method raises a `DOMException` object with its *code* set to the following:

`INVALID_CHARACTER_ERR` if *qualifiedName* contains an illegal character

`NAMESPACE_ERR` if *qualifiedName* is malformed or *qualifiedName* is prefixed with `xml` and *namespaceURI* is something other than `http://www.w3.org/XML/1998/namespace`

`WRONG_DOCUMENT_ERR` if *doctype* has already been used with another document or was created from a different DOM implementation

Example

Listing 10.40 uses a `DOMImplementation` object to create a new Document.

Listing 10.40 Creating a Document Using the `createDocument()` Method of the `DOMImplementation` Object

```
<html>
<script language="JScript">
<!--
var domImplObj = new DOMImplementation();
var documentObj = domImplObj.createDocument
    ("http://foobar.org/schema", "foo:bar", null);
//-->
</script>
</html>
```

DOMImplementation.createDocumentType()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOMImplementation.createDocumentType
➔(qualifiedName, publicId, systemId)

Description

The createDocumentType() method of the DOMImplementation object takes the three string arguments *qualifiedName*, *publicId*, and *systemId* and returns a new empty DocumentType object. This method raises a DOMException with its *code* set to INVALID_CHARACTER_ERR if *qualifiedName* contains an illegal character or NAME_SPACE_ERR if *qualifiedName* is malformed.

Example

Listing 10.41 uses a DOMImplementation object to create a new empty XML document type.

Listing 10.41 *Creating an Empty Document Type Using the createDocumentType() Method of the DOMImplementation Object*

```
<html>
<script language="JScript">
<!--
var domImplementation = new DOMImplementation();
var docTypeObj = domImplObj.createDocumentType("foo:bar",
"http://foobar.com/foo.xml", "http://foobar.com/bar.xml");
// -->
</script>
</html>
```

DOMImplementation.hasFeature()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOMImplementation.hasFeature(*feature*, *version*)

Description

The hasFeature() method of the DOMImplementation object takes the DOMString arguments *feature* and *version* and returns a Boolean value indicating whether the feature is available in the DOMImplementation implementation.

Example

Listing 10.42 uses a DOMImplementation object to check whether level 2.0 contains support for HTML Events.

Listing 10.42 Using the DOMImplementation.hasFeature() Method

```
<html>
<script language="JScript">
<!--
var domImplObj = new DOMImplementation();
var hasHTMLEvents = domImplObj.hasFeature("HTMLEvents", "2.0");
// -->
</script>
</html>
```

Element

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

The Element object represents an element in an XML or HTML document. An Element object inherits all methods and properties from Node. Table 10.7 lists all properties and methods of the Element object.

Table 10.7 Arguments Associated With Element Object

Type	Item	Description
Property	tagName	The name of this element.
	getAttribute()	Returns the attribute value by name.
Method	getAttributeNode()	Returns an attribute node by name.
	getAttributeNodeNS()	Returns an attribute node by namespace URI and local name.
	getAttributeNS()	Returns an attribute value using namespace URI and local name.
	getElementsByTagName()	Returns a NodeList of elements with the specified name.
	getElementsByTagNameNS()	Returns a NodeList of elements with the specified namespace URI and local name.
	hasAttribute()	Returns whether this element has a value for the named attribute.
	hasAttributeNS()	Returns whether this element has a value for an attribute indicated by the specified namespace URI and local name.

Type	Item	Description
	<code>removeAttribute()</code>	Removes the named attribute value from this element.
	<code>removeAttributeNS()</code>	Removes the attribute value with given namespace URI and local name.
	<code>removeAttributeNode()</code>	Removes the specified attribute node.
	<code>setAttribute()</code>	Sets an attribute with a given name to the specified value.
	<code>setAttributeNS()</code>	Sets an attribute with a given local name and namespace URI to the given value.
	<code>setAttributeNode()</code>	Sets a new attribute node.
	<code>setAttributeNodeNS()</code>	Adds a new attribute with the given namespace URI and local name.

Example

Listing 10.43 creates an `Element` from a `Document` object.

Listing 10.43 *Creating an Element from a Document*

```
<html>
<script language="JScript">
<!--
var newElementObj = documentObj.createElement("item");
// -->
</script>
</html>
```

Element.getAttribute()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getAttribute(name)`

Description

The `getAttribute()` method of the `Element` object returns the string value for the attribute *name*.

Example

Listing 10.44 gets a quantity attribute from an element representing a line item on a purchase order.

Listing 10.44 *Getting an Attribute from an Element Using the `getAttribute()` Method of the Element Object*

```
<html>
<script language="JScript">
<!--
var qtyAttrValue = elementObj.getAttribute("quantity");
// -->
</script>
</html>
```

Element.getAttributeNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getAttributeNode(name)`

Description

The `getAttributeNode()` method of the `Element` object retrieves the attribute specified by name and returns it as an `Attr`. If there is no such `Attr` object, *null* is returned.

Example

Listing 10.45 gets an attribute node from an element and sets its value.

Listing 10.45 *Getting an Attribute Node Using the `getAttributeNode()` Method of the Element Object*

```
<html>
<script language="JScript">
<!--
var qtyAttr = elementObj.getAttributeNode("quantity");
qtyAttr.value = "3";
// -->
</script>
</html>
```

Element.getAttributeNodeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getAttributeNodeNS(namespaceURI, localName)`

Description

The `getAttributeNodeNS()` method of the `Element` object returns the `Attr` value containing the specified *namespaceURI* and *localName*.

Example

Listing 10.46 retrieves an attribute node representing a quantity from an element representing a line item.

Listing 10.46 *Retrieving an Attribute Node Using the `getAttributeNodeNS()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
var qtyAttrNodeObj = invoiceElementObj.getAttributeNodeNS
    ("http://foo.com/namespace", "foo:quantity");
// -->
</script>
</html>
```

Element.getAttributeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getAttributeNS(namespaceURI, localName)`

Description

The `getAttributeNS()` method of the `Element` object returns the string value of the attribute specified by the given *namespaceURI* and *localName*.

Example

Listing 10.47 gets an element's attribute using namespace URI and local name.

Listing 10.47 *Getting an Attribute Using Namespace URI Using the `getAttributeNS()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
var qtyAttrValue = elementObj.getAttributeNS
    ("http://foo.com/namespace", "foo:quantity");
// -->
</script>
</html>
```

Element.getElementsByTagName()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getElementsByTagName(name)`

Description

The `getElementsByTagName()` method of the `Element` object returns a `NodeList` containing all `Elements` named *name*.

Example

Listing 10.48 gets a list of line items from an element representing an invoice.

Listing 10.48 *Getting Elements by Tag Name Using the `getElementsByTagName()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
var lineItemList = invoiceElementObj.getElementsByTagName("lineItem");
// -->
</script>
</html>
```

Element.getElementsByTagNameNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.getElementsByTagNameNS(namespaceURI, localName)`

Description

The `getElementsByTagNameNS()` method of the `Element` object returns a `NodeList` of all descendant `Elements` with the given *namespaceURI* and *localName*.

Example

Listing 10.49 gets a list of line items from an element representing an invoice.

Listing 10.49 *Getting Elements by Tag name Using the `getElementsByTagNameNS()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
```

```
var lineItemList = invoiceElementObj.getElementsByTagNameNS
  ("http://foo.com/namespace", "lineItem");
// -->
</script>
</html>
```

Element.hasAttribute()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.hasAttribute(name)`

Description

The `hasAttribute()` method of the `Element` object returns `true` if this element has an attribute named *name*.

Example

Listing 10.50 determines whether a line item element has an attribute for price set.

Listing 10.50 *Checking Whether an Element Has an Attribute Using the `hasAttribute()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
if(!lineItemElementObj.hasAttribute("price"))
    lineItemElementObj.setAttribute("price", "3.44");
// -->
</script>
</html>
```

Element.hasAttributeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.hasAttributeNS(namespaceURI, localName)`

Description

The `hasAttributeNS()` method of the `Element` object returns `True` if there exists an attribute of this element that has the specified *namespaceURI* and *localName*.

Example

Listing 10.51 determines whether a line item element has an attribute for price set.

Listing 10.51 *Checking Whether an Element Has an Attribute Using the `hasAttributeNS()` Method of the Element Object*

```
<html>
<script language="JavaScript">
<!--
if(!lineItemElementObj.hasAttributeNS("http://foo.com/namespace", "price"))
    lineItemElementObj.setAttribute("price", "3.44");
// -->
</script>
</html>
```

Element.removeAttribute()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.removeAttribute(name)`

Description

The `removeAttribute()` method of the `Element` object removes the named attribute specified by the string *name* from this element.

Example

Listing 10.52 removes the quantity attribute from an element representing a line item.

Listing 10.52 *Removing an Attribute Using the `removeAttribute()` Method of the Element Object*

```
<html>
<script language="JavaScript">
<!--
elementObj.removeAttribute("quantity");
// -->
</script>
</html>
```

Element.removeAttributeNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.removeAttributeNode(oldAttr)`

Description

The `removeAttributeNode()` method of the `Element` object removes *oldAttr*. A `DOMException` object is raised with the value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only or with value `NOT_FOUND_ERR` if *oldAttr* isn't an attribute of this element.

Example

Listing 10.53 Removes an attribute node from an element.

Listing 10.53 Removing an Attribute Node Using the `removeAttributeNode()` Method of the `Element` Object

```
<html>
<script language="JScript">
<!--
elementObj.removeAttributeNode(oldNode);
// -->
</script>
</html>
```

Element.removeAttributeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`elementObj.removeAttributeNS(namespaceURI, localName)`

Description

The `removeAttributeNS()` method of the `Element` object removes the attribute specified by the given *namespaceURI* and *localName*. A `DOMException` object can be raised with value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.54 removes the quantity attribute from an element representing an invoice.

Listing 10.54 Removing an Attribute Using the `removeAttributeNS()` Method of the `Element` Object

```
<html>
<script language="JScript">
<!--
invoiceElementObj.removeAttributeNS
    ("http://foo.com/namespace", "foo:quantity");
// -->
</script>
</html>
```

Element.setAttribute()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
elementObj.setAttribute(name, value)
```

Description

The `setAttribute()` method of the `Element` object adds a new attribute named *name* to *value*. If the attribute is already present, its value is changed to *value*. A `DOMException` object can be raised with value `INVALID_CHARACTER_ERR` if the specified *name* contains an illegal character or with a value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.55 sets a quantity attribute for an element that represents a line item.

Listing 10.55 *Setting an Attribute for an Element Using the `setAttribute()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
elementObj.setAttribute("quantity", "3");
// -->
</script>
</html>
```

Element.setAttributeNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
elementObj.setAttributeNode(newAttr)
```

Description

The `setAttributeNode()` method of the `Element` object adds a new attribute node *newAttr*. The newly placed `Attr` is returned. If *newAttr* is already an attribute of this element, `Null` is returned.

Example

Listing 10.56 sets a attribute node representing a quantity for an element representing a line item.

Listing 10.56 *Setting an Attribute Node Using the* `setAttributeNode()` *Method of the* `Element` *Object*

```
<html>
<script language="JScript">
<!--
var qtyAttr = documentObj.createAttribute("quantity");
qtyAttr.value = "3";
elementObj.setAttributeNode(qtyAttr);
// -->
</script>
</html>
```

Element.setAttributeNodeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
elementObj.setAttributeNodeNS(newAttr)
```

Description

The `setAttributeNodeNS()` method of the `Element` object adds a new attribute *newAttr*. If an attribute with matching local name and namespace URI already exists, it will be replaced. A `DOMException` object can be raised with value *WRONG_DOCUMENT_ERR* if *newAttr* was created by a document other than the one that created this element or with value *INUSE_ATTRIBUTE_ERR* if *newAttr* is already an attribute of another element. In this case, the *newAttr* needs to be cloned for reuse.

Example

Listing 10.57 sets a new quantity attribute for an element representing a line item.

Listing 10.57 *Setting an Attribute Using the* `setAttributeNodeNS()` *Method of the* `Element` *Object*

```
<html>
<script language="JScript">
<!--
var qtyAttrObj = documentObj.createAttribute("quantity");
lineItemElementObj.setAttributeNodeNS(qtyAttrObj);
// -->
</script>
</html>
```

Element.setAttributeNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
elementObj.setAttributeNS(namespaceURI, qualifiedName, value)
```

Description

The `setAttributeNS()` method of the `Element` object adds a new attribute to this element. If an attribute with the same *namespaceURI* and *qualifiedName* already exists, its value will be replaced with the new *value*.

Example

Listing 10.58 sets an attribute using its namespace URI.

Listing 10.58 *Setting an Attribute Using the `setAttributeNS()` Method of the `Element` Object*

```
<html>
<script language="JScript">
<!--
invoiceElementObj.setAttributeNS
    ("http://foo.com/namespace", "foo:quantity", "4");
// -->
</script>
</html>
```

Element.tagName

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
elementObj.tagName
```

Description

The read-only property `tagName` of the `Element` object returns the name of this element.

Example

Listing 10.59 queries an `Element` for its *tagName*.

Listing 10.59 *Reading the `tagName` Property of the `Element` Object*

```
<html>
<script language="JScript">
<!--
```

```
var tagName = elementObj.tagName;  
// -->  
</script>  
</html>
```

Entity

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

The Entity object represents an XML entity that is either parsed or unparsed. It inherits all methods and properties of Node. Table 10.8 lists the properties for the Entity object.

Table 10.8 *Arguments Associated With Entity Object*

Type	Item	Description
Property	notationName	Returns the name of the notation for this entity.
	publicId	The public identifier of this entity.
	systemId	The system identifier of this entity

Example

An Entity is instantiated using the `createEntity()` method of the Document object. Entity inherits all methods and properties from the Node object.

Entity.notationName

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`entityObj.notationName`

Description

The read-only property `notationName` of the Entity object returns the name of the notation for this entity if it is unparsed. If the entity is parsed, this returns *null*.

Example

Listing 10.60 retrieves the notation name from a Entity object.

Listing 10.60 *Reading the notationName Property of the Entity Object*

```
<html>
<script language="JScript">
<!--
var notationName = entityObj.notationName;
// -->
</script>
</html>
```

Entity.publicId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

entityObj.publicId

Description

The read-only property `publicId` of the `Entity` object returns the public identifier associated with this entity. If it isn't specified, *null* is returned.

Example

Listing 10.61 retrieves the public identifier from an `Entity` object.

Listing 10.61 *Reading the publicId Property of the Entity Object*

```
<html>
<script language="JScript">
<!--
var publicId = entityObj.publicId;
// -->
</script>
</html>
```

Entity.systemId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

entityObj.systemId

Description

The read-only property `systemId` of the `Entity` object returns the system identifier associated with this entity. If the system ID isn't specified, this returns *null*.

Example

Listing 10.62 retrieves the system identifier from an Entity object.

Listing 10.62 *Reading the `systemId` Property of the Entity Object*

```
<html>
<script language="JScript">
<!--
var systemId = entityObj.systemId;
// -->
</script>
</html>
```

EntityReference

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

An EntityReference object inherits all methods and properties from Node.

Example

An EntityReference is instantiated using the `createEntityReference()` method from the Document object.

NamedNodeMap

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object

Description

This is a list of DOM nodes with names associated with each. Table 10.9 lists all properties and methods of the NamedNodeMap object.

Table 10.9 *Arguments Associated With NamedNodeMap Object*

Type	Item	Description
Property	<code>length</code>	This read-only property returns the number of nodes in this map.
Method	<code>getNamedItem()</code>	Returns a node specified by a name.
	<code>getNamedItemsNS()</code>	Returns a node specified by local name and namespace.

Table 10.9 *Continued*

Type	Item	Description
	<code>item()</code>	Returns the node referenced by the specified index.
	<code>removeNamedItem()</code>	Removes the node specified by a name.
	<code>removeNamedItemNS()</code>	Removes the node specified by local name and namespace URI.
	<code>setNamedItem()</code>	Adds a node using its <code>nodeName</code> attribute for the mapped name.
	<code>setNamedItemNS()</code>	Adds a node using its namespace URI and local name for the mapped name.

Example

Listing 10.63 retrieves the `NamedNodeList` `attributes` property from a `HTMLBodyElement` object and reads its `bgColor` property.

Listing 10.63 *Getting the attributes property as a `NamedNodeMap` and getting the `bgColor` property.*

```
<html>
<script language="JScript">
<!--
var bodyAtts = bodyObj.attributes;
var color = bodyAtts.bgColor;
// -->
</script>
</html>
```

NamedNodeMap.getNamedItem()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
namedNodeMapObj.getNamedItem(name)
```

Description

The `getNamedItem()` method of the `NamedNodeMap` object returns a node specified by *name*.

Example

Listing 10.64 retrieves an attribute object by name from a `NamedNodeMap` and processes it.

Listing 10.64 Retrieving an Item by Name Using the `getNamedItem()` Method of `NamedNodeMap`

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var size = attrList.getNamedItem("size");
// -->
</script>
</html>
```

NamedNodeMap.getNamedItemNS()

JavaScript 1.5+, JScript 5.0+
 Nav6+, IE5+

Syntax

`namedNodeMap.getNamedItemNS(namespaceURI, localName)`

Description

The `getNamedItemNS()` method of the `NamedNodeMap` object returns the node specified by *localName* and *namespaceURI*. This method returns `Null` if no node exists for the specified *localName* or *namespaceURI*.

Example

Listing 10.65 retrieves an attribute object by *localName* and a namespace URI from a `NamedNodeMap` and processes it.

Listing 10.65 Retrieving an Item by Name Using the `getNamedItemNS()` Method of the `NamedNodeMap` Object

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var size = attrList.getNamedItemNS("http://foo.com/namespace", "foo:size");
// -->
</script>
</html>
```

NamedNodeMap.item()

JavaScript 1.5+, JScript 5.0+
 Nav6+, IE5+

Syntax

`namedNodeMapObj.item(index)`

Description

The `item()` method of the `NamedNodeMap` object returns the node in position *index* in this map. If no node exists at this position, *null* is returned.

Example

Listing 10.66 iterates through an attribute list processing each one.

Listing 10.66 *Pulling an Item at a Certain Index Using the `item()` Method of the `NamedNodeMap` Object*

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var i = 0;
while(i < attrList.length) {
    processAttr(attrList.item(i));
    i++;
}
// -->
</script>
</html>
```

NamedNodeMap.length

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`namedNodeMapObj.length`

Description

The `length` property of the `NamedNodeMap` object returns the number of nodes contained in this named node map.

Example

Listing 10.67 checks to see whether a node has any attributes. If it does, the attributes are iterated over and processed. The *attrList* is an example of a `NamedNodeMap`.

Listing 10.67 *Testing Against `NamedNodeMap` length*

```
<html>
<script language="JScript">
<!--
if(nodeObj.hasAttributes()) {
    var attrList = nodeObj.attributes;
    var i = 0;
    while(i < attrList.length) {
        processAttr(attrList.Item(i));
    }
}
```



```

        i++;
    }
}
// -->
</script>
</html>

```

NamedNodeMap.removeNamedItem()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`namedNodeMapObj.removeNamedItem(name)`

Description

The `removeNamedItem()` method of the `NamedNodeMap` object removes the node specified by *name* and returns it. A `DOMException` can be raised with the value `NOT_FOUND_ERR` if no *name* node is in this map, or with value `NO_MODIFICATION_ALLOWED_ERR` if this map is read-only.

Example

Listing 10.68 removes a named item from an attribute list.

Listing 10.68 *Removing a Named Item Using the* `removeNamedItem()` *Method of the* `NamedNodeMap` *Object*

```

<html>
<script language="JScript">
<!--
attrList.removeNamedItem("size");
// -->
</script>
</html>

```

NamedNodeMap.removeNamedItemNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`namedNodeMapObj.removeNamedItemNS(name)`

Description

The `removeNamedItemNS()` method of the `NamedNodeList` object removes a node specified by *name* and returns it. If no node exists with the name specified, *null* is returned.

Example

Listing 10.69 removes an item from an attribute list.

Listing 10.69 Removing an Item Using the `removeNamedItem()` Method of the `NamedNodeMap` Object

```
<html>
<script language="JScript">
<!--
attrListObj.removeNamedItemNS("size");
// -->
</script>
</html>
```

NamedNodeMap.setNamedItem()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nameNodeMapObj.setNamedItem(node)`

Description

The `setNamedItem()` method of the `NamedNodeMap` object adds a *node* using its *nodeName* property. If a node is already present in this named node map with the same name, it will be replaced. The newly added node is returned. A `DOMException` object can be raised with one of the following values:

WRONG_DOCUMENT_ERR if the node was created from a different document than the one that created this map

NO_MODIFICATION_ALLOWED_ERR if this map is read-only

INUSE_ATTRIBUTE_ERR if node is an `Attr` that is already an attribute of another `Element` object. If this is the case, the DOM user must clone the `Attr` for reuse

Example

Listing 10.70 sets a new named attribute in an attribute list.

Listing 10.70 Adding a New Named Attribute to an Attribute List Using the `setNamedItem()` Method of `NamedNodeMap`

```
<html>
<script language="JScript">
<!--
var newAttr = documentObj.createAttribute("size");
attrList.setNamedItem(newAttr);
// -->
</script>
</html>
```

NamedNodeMap.setNamedItemNS()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`namedNodeMapObj.setNamedItemNS(node)`

Description

The `setNamedItemNS()` method of the `NamedNodeMap` object adds a *node* to the map using its *namespaceURI* and *localName* properties as keys on the map. If a node already exists at the specified keys, it is replaced. A `DOMException` object can be raised with one of the following values:

WRONG_DOCUMENT_ERR if the node was created by a document other than the one that created this map

NO_MODIFICATION_ALLOWED_ERR if this map is read-only

INUSE_ATTRIBUTE_ERR if node is an `Attr` that is already an attribute of another `Element` object. In this case, the DOM user must explicitly clone the `Attr` node for reuse

Example

Listing 10.71 sets a new named attribute in an attribute list.

Listing 10.71 Adding a New Named Attribute to an Attribute List Using the `setNamedItemNS()` Method of the `NamedNodeMap` Object

```
<html>
<script language="JScript">
<!--
var newAttr = documentObj.createAttribute("size");
attrList.setNamedItemNS(newAttr);
// -->
</script>
</html>
```

Node

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

The `Node` object represents a node of an XML document. Table 10.10 lists all constants, properties, and methods of the `Node` object.

Table 10.10 Arguments Associated With Node Object

Type	Item	Description
Constant	ELEMENT_NODE	Represents the type of this node when it is instantiated as an Element. Numeric value is 1.
	ATTRIBUTE_NODE	Represents the type of this node when it is instantiated as an Attr. Numeric value is 2.
	TEXT_NODE	Represents the type of this node when it is instantiated as Text. Numeric value is 3.
	CDATA_SECTION_NODE	Represents the type of this node when it is instantiated as a CDATASection object. Numeric value is 4.
	ENTITY_REFERENCE_NODE	Represents the type of this node when it is instantiated as an EntityReference object. Numeric value is 5.
	ENTITY_NODE	Represents the type of this node when it is instantiated as an Entity object. Numeric value is 6.
	PROCESSING_INSTRUCTION_NODE	Represents the type of this node when it is instantiated as ProcessingInstruction object. Numeric value is 7.
	COMMENT_NODE	Represents the type of this node when it is instantiated as an Comment object. Numeric value is 8.
	DOCUMENT_NODE	Represents the type of this node when it is instantiated as a Document object. Numeric value is 10.
	DOCUMENT_TYPE_NODE	Represents the type of this node when it is instantiated as a DocumentType object. Numeric value is 10.
	DOCUMENT_FRAGMENT_NODE	Represents the type of this node when it is instantiated as a DocumentFragment object. Numeric value is 11.
	NOTATION_NODE	Represents the type of this node when it is instantiated as a Notation object. Numeric value is 12.

Type	Item	Description
Property	attributes	Returns a NamedNodeMap containing all attributes of this node.
	childNodes	Returns a NodeList containing all children of this node.
	firstChild	Returns a Node object containing the first child of this node.
	lastChild	Returns a Node object containing the last child of this node.
	localName	Returns a DOMString object containing the local part of the qualified name for this node.
	namespaceURI	Returns a DOMString containing the namespace URI.
	nextSibling	Returns the node immediately following this node.
	nodeName	Returns a DOMString containing the name of this node depending on its type.
	nodeType	Returns a constant unsigned short representing the type of this node.
	nodeValue	Returns a DOMString object containing the value of this node depending on its type.
	ownerDocument	Returns a Document object associated with this node.
	parentNode	Returns a Node object that serves as parent to this node.
	prefix	Returns a DOMString object containing the namespace prefix for this node.
	previousSibling	Returns the node immediately before this node.
Method	appendChild()	Returns a newly added child of this node.
	cloneNode()	Returns a new node that is a clone of this node.
	hasAttributes()	Returns whether this node has any attributes.
	hasChildNodes()	Returns whether this node has any children.
	insertBefore()	Returns the newly inserted node before this node.

Table 10.10 Continued

Type	Item	Description
	isSupported()	Returns whether a certain feature is supported in this DOM implementation.
	normalize()	Attempts to normalize the contents of this node.
	removeChild()	Returns a node representing a child that has been removed from this node.
	replaceChild()	Replaces a child node with a new node and returns the new node.

Example

Because Node is a parent object to all DOM Core objects, it is never instantiated on its own. Instead, its methods and properties are accessed via its child objects.

Node.attributes

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

nodeObj.attributes

Description

The read-only property attributes of the Node object returns a NamedNodeMap containing all attributes of this node.

Example

Listing 10.72 iterates through a list of node attributes and processes them.

Listing 10.72 Iterating Through the attributes Property of the Node Object

```
<html>
<script language="JScript">
<!--
var attrList = nodeObj.attributes;
var i = 0;
while(i < attrList.length) {
    processAttr(attrList.item(i);
    i++;
}
// -->
</script>
</html>
```

Node.appendChild()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.appendChild(newChild)`

Description

The `appendChild()` method of the `Node` object adds *newChild* to the list of children for this node. If *newChild* is already in the tree, it is first removed. The newly added child node is returned. A `DOMException` can be raised with the value `HIERARCHY_REQUEST_ERR` if this node is of a type that doesn't accept children of the *newChild* type, with value `WRONG_DOCUMENT_ERR` if *newChild* was created by a document other than the one that created this node, or with value `NO_MODIFICATION_ALLOWED` if this node is read-only.

Example

Listing 10.73 adds a new child to a node.

Listing 10.73 Appending a Child Using the `appendChild()` Method of the `Node` Object

```
<html>
<script language="JScript">
<!--
var newElementObj = documentObj.createElement("item");
nodeObj.appendChild(newElementObj);
// -->
</script>
</html>
```

Node.childNodes

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.childNodes`

Description

The `childNodes` attribute of the `Node` object returns a `NodeList` containing all children of this node.

Example

Listing 10.74 returns all children of a node and then iterates through the list, printing out the text contents if the node is a text node.

Listing 10.74 *Iterating Through the `childNodes` Property of the Node Object*

```
<html>
<script language="JScript">
<!--
var nodeListObj = nodeObj.childNodes;
var i = 0;
while(i < nodeListObj.Length) {
    if(nodeListObj.item(i).nodeType == Node.TEXT_NODE)
        handleTextNode(nodeListObj.Item(i));
    i++;
}
// -->
</script>
</html>
```

Node.cloneNode()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
nodeObj.cloneNode(deep)
```

Description

The `cloneNode()` method of the Node object returns a duplicate copy of this node. If *deep* is true, the subtree under this node is cloned along with this node.

Example

Listing 10.75 clones a node and then adds the clone as a child to another node.

Listing 10.75 *Cloning a Node Using the `cloneNode()` Method of the Node Object*

```
<html>
<script language="JScript">
<!--
var clonedNodeObj = nodeObj.cloneNode(true);
anotherNodeObj.appendChild(clonedNodeObj);
// -->
</script>
</html>
```

Node.firstChild

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.firstChild`

Description

The read-only property `firstChild` of the `Node` object returns the first child of this node.

Example

Listing 10.76 checks to see the type of the first child of this node.

Listing 10.76 *Reading the `firstChild` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
var childNodeObj = nodeObj.firstChild;
if(childNodeObj.nodeName == "foo")
    process(childNodeObj);
// -->
</script>
</html>
```

Node.hasAttributes()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.hasAttributes()`

Description

The `hasAttributes()` method of the `Node` object returns `True` if this node has any attributes attached to it.

Example

Listing 10.77 checks to see whether a node has any attributes and iterates the list if it does.

Listing 10.77 *Checking for Attributes Using the `hasAttributes()` Method of the `Node` Object*

```
<html>
<script language="JScript">
<!--
if(nodeObj.hasAttributes()) {
    var attrList = nodeObj.attributes;
    var i = 0;
```

Listing 10.77 Continued

```
        while(i < attrList.Length) {
            processAttr(attrList.Item(i);
            i++;
        }
    }
}
// -->
</script>
</html>
```

Node.hasChildNodes()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.hasChildNodes()`

Description

The `hasChildNodes()` method of the `Node` object returns *true* if this node has children.

Example

Listing 10.78 tests to see if a node has children.

Listing 10.78 Checking for children using the `hasChildNodes()` method of the `Node` object

```
<html>
<script language="JScript">
<!--
if(nodeObj.hasChildNodes())
    handleChildren(nodeObj);
// -->
</script>
</html>
```

Node.insertBefore()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.insertBefore(newChild, refChild)`

Description

The `insertBefore()` method of the `Node` object inserts the node *newChild* before the existing child node *refChild*. If *newChild* is already in the tree, it is first removed. The

newly inserted child node is returned. A `DOMException` object can be raised with one of the following values:

HIERARCHY_REQUEST_ERR if this node is a type that doesn't allow children of the *newChild* type

WRONG_DOCUMENT_ERR if the *newChild* was created from a different document than the one that created this node

NO_MODIFICATION_ALLOWED_ERR if this node is read-only

NOT_FOUND_ERR if *refChild* isn't a child of this node

Example

Listing 10.79 inserts a new child node before a reference child node.

Listing 10.79 *Inserting a New Child Node Using the* `insertBefore()` *Method of the* `Node` *Object*

```
<html>
<script language="JScript">
<!--
var newElementObj = documentObj.createElement("item");
nodeObj.insertBefore(newElementObj, existingChild);
// -->
</script>
</html>
```

Node.isSupported()

JavaScript 1.5+, JScript 5.0+

Nav6+, IE5+

Syntax

```
nodeObj.isSupported(feature, version)
```

Description

The `isSupported()` method of the `Node` object returns whether the specified *feature* and *version* are supported by this node.

Example

Listing 10.80 queries a node for the availability of a certain feature.

Listing 10.80 *Checking Whether a Feature Is Supported Using the* `isSupported()` *Method of the* `Node` *Object*

```
<html>
<script language="JScript">
<!--
if(nodeObj.isSupported("HTMLEvents", "1.0")
    handleHTMLEvents();
```

Listing 10.80 Continued

```
// -->
</script>
</html>
```

Node.lastChild

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.lastChild`

Description

The read-only property `lastChild` of the `Node` object returns the last child of this node.

Example

Listing 10.81 checks to see the type of the last child of this node.

Listing 10.81 Reading the `lastChild` Property of the `Node` Object

```
<html>
<script language="JScript">
<!--
var childNodeObj = nodeObj.lastChild;
if(childNodeObj.nodeName == "foo")
    process(childNodeObj);
// -->
</script>
</html>
```

Node.localName

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.localName`

Description

The read-only property `localName` of the `Node` object returns the local part of the qualified name for this node.

Example

Listing 10.82 takes the local name of a node and prepends the prefix to give the qualified name of the node.

Listing 10.82 *Assigning the localName Property of the Node Object*

```
<html>
<script language="JScript">
<!--
var localName = nodeObj.localName;
var prefix = nodeObj.prefix;
var qualifiedName = prefix+": "+localName;
// -->
</script>
</html>
```

Node.namespaceURI

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

nodeObj.namespaceURI

Description

The read-only property namespaceURI of the Node object returns a string containing the namespace URI for this node.

Example

Listing 10.83 checks the prefix against the namespace URI and throws an exception if the prefix is foo and the namespace URI is not http://foo.com/namespace.

Listing 10.83 *Reading the namespaceURI Property of the Node Object*

```
<html>
<script language="JScript">
<!--
if((nodeObj.Prefix == "foo" )
    && (nodeObj.NamespaceURI != "http://foo.com/namespace"))
    throw new DOMException(DOMException.NAMESPACE_ERR);
// -->
</script>
</html>
```

Node.nextSibling

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

nodeObj.nextSibling

Description

The read-only property `nextSibling` of the `Node` object returns the node immediately after this node.

Example

Listing 10.84 checks to see the type of the next sibling of this node.

Listing 10.84 *Reading the `nextSibling` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
var siblingNodeObj = nodeObj.nextSibling;
if(siblingNodeObj.nodeName == "foo")
    process(siblingNodeObj);
// -->
</script>
</html>
```

Node.nodeName

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.nodeName`

Description

The `nodeName` property of the `Node` object returns the name of this node as indicated by the *nodeType* attribute.

Example

Listing 10.85 takes a `Node` and performs a particular operation depending on the `nodeName`.

Listing 10.85 *Checking the `nodeName` Property*

```
<html>
<script language="JScript">
<!--
if(elementObj.nodeName == "foobar")
    processElement(elementObj);
// -->
</script>
</html>
```

Node.nodeType

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.nodeType`

Description

The `nodeType` property of the `Node` object returns a number indicating the node type of this node as represented by the constants listed previously.

Example

Listing 10.86 checks the type of a node and processes it accordingly.

Listing 10.86 *Checking the `nodeType` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
var nodeType = textObj.nodeType;
if(nodeType == Node.TEXT_NODE)
    handleText(textObj);
// -->
</script>
</html>
```

Node.nodeValue

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.nodeValue`

Description

The `nodeValue` property of the `Node` object returns the value of this node depending on its `nodeType` attribute

Example

Listing 10.87 illustrates iterating through a list of elements processing each object's `nodeValue` property.

Listing 10.87 *Processing the `nodeValue` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
```

Listing 10.87 Continued

```
var elementListObj = documentObj.getElementsByTagName
    ("http://foo.com/namespace", "foo:item");
var i = 0;
while ( i < elementListObj.length) {
    processElementValue(elementListObj.item(i).nodeValue);
}
// -->
</script>
</html>
```

Node.normalize()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
nodeObj.normalize()
```

Description

The `normalize()` method of the `Node` object puts all `Text` nodes in the full depth of the subtree underneath this node. This can be used to ensure that the DOM view of the document is the same as if it were saved and reloaded.

Example

Listing 10.88 demonstrates normalizing a node before saving it to a file.

Listing 10.88 Normalizing a Node Using the `normalize()` Method of the `Node` Object

```
<html>
<script language="JScript">
<!--
nodeObj.normalize();
writeNode(nodeObj);
// -->
</script>
</html>
```

Node.ownerDocument

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
nodeObj.ownerDocument
```


Description

The read-only property `ownerDocument` of the `Node` object returns the `Document` object associated with this node.

Example

Listing 10.89 retrieves the owner document of this node and creates a new element.

Listing 10.89 *Reading the `ownerDocument` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
var documentObj = nodeObj.ownerDocument;
var elementObj = documentObj.createElement("lineItem");
nodeObj.appendChild(elementObj);
// -->
</script>
</html>
```

Node.parentNode

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.parentNode`

Description

The `parentNode` property of the `Node` object returns the parent of this node.

Example

Listing 10.90 checks to see whether a node object has a parent.

Listing 10.90 *Checking the `parentNode` Property of the `Node` Object*

```
<html>
<script language="JScript">
<!--
if(elementObj.parentNode != null)
    processAsChild(elementObj);
// -->
</script>
</html>
```

Node.prefix

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.prefix`

Description

The `prefix` property of the `Node` object returns the namespace prefix for this node. A `DOMException` object can be raised with one of the following values:

`INVALID_CHARACTER_ERR` if the prefix contains an illegal character

`NO_MODIFICATION_ALLOWED_ERR` if the node is read-only

`NAMESPACE_ERR` if the prefix is malformed, the `NamespaceURI` is `Null`, or the prefix is `xml` and the `NamespaceURI` isn't `http://www.w3.org/XML/1998/namespace`

Example

Listing 10.91 checks the prefix against the namespace URI and throws an exception if the prefix is `foo` and the namespace URI isn't `http://foo.com/namespace`.

Listing 10.91 *Reading the prefix Property of the Node Object*

```
<html>
<script language="JScript">
<!--
if((nodeObj.prefix == "foo")
    && (nodeObj.namespaceURI != "http://foo.com/namespace"))
    throw new DOMException(DOMException.NAMESPACE_ERR);
// -->
</script>
</html>
```

Node.previousSibling

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`nodeObj.previousSibling`

Description

The read-only property `previousSibling` of the `Node` object returns the node immediately before this node.

Example

Listing 10.92 illustrates checking to see the type of the previous sibling of this node.

Listing 10.92 *Reading the previousSibling Property of the Node Object*

```
<html>
<script language="JScript">
<!--
var siblingNodeObj = nodeObj.previousSibling;
if(siblingNodeObj.nodeName == "foo")
    process(siblingNodeObj);
// -->
</script>
</html>
```

Node.removeChild()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
nodeObj.removeChild(oldChild)
```

Description

The `removeChild()` method of the `Node` object removes the child node specified by *oldChild* from the list of children of this node. A `DOMException` object can be raised with value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only, or with value `NOT_FOUND_ERR` if *oldChild* isn't a child of this node.

Example

Listing 10.93 removes a child from a node.

Listing 10.93 *Removing a Child from a Node Using the removeChild() Method of the Node Object*

```
<html>
<script language="JScript">
<!--
nodeObj.removeChild(existingChild);
// -->
</script>
</html>
```

Node.replaceChild()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
nodeObj.replaceChild(newChild, oldChild)
```

Description

The `replaceChild()` method of the `Node` object replaces the child node *oldChild* with a new child node *newChild*. If *newChild* is already in the tree, it is first removed. A `DOMException` object is raised with one of the following values:

HIERARCHY_REQUEST_ERR if this node is of a type that doesn't allow children of the *newChild* type

WRONG_DOCUMENT_ERR if the *newChild* was created from a different `Document` than the one that created this node

NO_MODIFICATION_ALLOWED_ERR if this node or the parent of the new node is read-only

NOT_FOUND_ERR if the *oldChild* isn't a child of this node

Example

Listing 10.94 replaces an existing child node with a new node.

Listing 10.94 Replacing a Child Node Using the `replaceChild()` Method of the `Node` Object

```
<html>
<script language="JScript">
<!--
var newElementObj = documentObj.createElement("item");
nodeObj.replaceChild(newElementObj, existingChild);
// -->
</script>
</html>
```

Notation

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

`Notation` inherits all properties and methods from its parent class `Node`. A `Notation` declares an unparsed entity in an XML document. Table 10.11 lists all properties of the `Notation` object.

Table 10.11 Arguments Associated With `Notation` Object

Type	Item	Description
Property	<code>publicId</code>	The public identifier of this notation
	<code>systemId</code>	The system identifier of this notation

Example

A Notation is instantiated using the `createNotation()` method of the Document object.

Notation.publicId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`notationObj.publicId`

Description

The read-only property `publicId` of the Notation object returns the public identifier of this notation. If the *PublicId* is not specified for this notation, it returns *null*.

Example

Listing 10.95 retrieves the public identifier from a Notation object.

Listing 10.95 *Reading the `publicId` Property of the Notation Object*

```
<html>
<script language="JScript">
<!--
var publicId = docTypeObj.publicId;
// -->
</script>
</html>
```

Notation.systemId

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`notationObj.systemId`

Description

The read-only property `systemId` of the Notation object returns the system identifier of this notation. If the *systemId* isn't specified for this notation, it returns *null*.

Example

Listing 10.96 retrieves the system identifier from a Notation object.

Listing 10.96 *Reading the `systemId` Property of the `Notation` Object*

```
<html>
<script language="JScript">
<!--
var systemId = docTypeObj.systemId;
// -->
</script>
</html>
```

ProcessingInstruction

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

`ProcessingInstruction` inherits all properties and methods from `Node`. `ProcessingInstruction` represents an XML processing instruction that is used to keep parser specific information in the text of the XML document. Table 10.12 lists the properties of the `ProcessingInstruction` object.

Table 10.12 *Arguments Associated With `ProcessingInstruction` Object*

Type	Item	Description
Property	<code>data</code>	The content of the processing instruction
	<code>target</code>	The target of this processing instruction

Example

A `ProcessingInstruction` is instantiated using the `createProcessingInstruction()` method of the `Document` object.

ProcessingInstruction.data

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`processingInstructionObj.data`

Description

The read-only property `data` of the `ProcessingInstruction` object returns the content of the processing instruction. A `DOMException` object can be raised with the value `NO_MODIFICATION_ALLOWED_ERR` when this node is read-only.

Example

Listing 10.97 illustrates the retrieval of the `data` property.

Listing 10.97 *Reading the `data` Property of the `ProcessingInstruction` Object*

```
<html>
<script language="JScript">
<!--
var instructionData= processingInstructionObj.data;
// -->
</script>
</html>
```

ProcessingInstruction.target

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`processingInstructionObj.target`

Description

The read-only property `target` of the `ProcessingInstruction` object returns the target of this processing instruction.

Example

Listing 10.97 illustrates the retrieval of the `target` property.

Listing 10.97 *Reading the `target` Property of the `processingInstruction` Object*

```
<html>
<script language="JScript">
<!--
var targetToken = processingInstructionObj.target;
// -->
</script>
</html>
```

Text

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

Core DOM object.

Description

A `Text` object represents textual content in an XML document. If no markup is present in an element's content, it is stored as a `Text` object. `Text` inherits all methods and properties of `CharacterData`, which, in turn, inherits from `Node`. An argument associated with the `Text` object is as follows:

<i>Type</i>	<i>Item</i>	<i>Description</i>
Method	<code>splitText()</code>	Breaks this node into two nodes at a specified offset

Example

A `Text` object is created by the `createTextNode()` method of the `Document` object.

Text.splitText()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`textObj.splitText(offset)`

Description

The `splitText()` method of the `Text` object breaks this text node into two nodes at the given *offset*. After the text is split, the new node becomes a sibling of this node and will contain all the text after the *offset*. The new `Text` node is returned. A `DOMException` can be raised with the value `INDEX_SIZE_ERR` if the specified *offset* is negative or greater than the size of the contained text data or with the value `NO_MODIFICATION_ALLOWED_ERR` if this node is read-only.

Example

Listing 10.98 demonstrates splitting a `Text` node's Data into another `Text` node.

Listing 10.98 Splitting Text Using the `splitText()` Method of the `Text` Object

```
<html>
<script language="JScript">
<!--
var textObj = documentObject.createTextNode("This is some text");
var splitTextObj = textObj.splitText(5);
```



```
// -->  
</script>  
</html>
```




CHAPTER 11

DOM HTML

This chapter extends the DOM Level 2 Core API to describe objects and methods specific to HTML documents. In general, the functionality needed to manipulate hierarchical document structures, elements, and attributes will be found in the core section; functionality that depends on the specific elements defined in HTML will be found in this chapter.

The goals of the HTML-specific DOM API are

- To specialize and add functionality that relates specifically to HTML documents and elements.
- To address issues of backward compatibility with the DOM Level 0.
- To provide convenience mechanisms, where appropriate, for common and frequent operations on HTML documents.

The key differences between the core DOM and the HTML application of DOM is that the HTML Document Object Model exposes a number of convenience methods and properties that are consistent with the existing DOM Level 1 and 0 models and are more appropriate to script writers. In many cases, these enhancements aren't applicable to a general DOM because they rely on the presence of a predefined DTD. The transitional and frameset DTDs for HTML 4.0 are assumed. Interoperability between implementations is only guaranteed for elements and attributes that are specified in the HTML 4.0 DTDs.

More specifically, this document includes the following specializations for HTML:

- An `HTMLDocument` interface, derived from the core `Document` interface. `HTMLDocument` specifies the operations and queries that can be made on a HTML document.

- An `HTMLElement` interface, derived from the core `Element` interface. `HTMLElement` specifies the operations and queries that can be made on any HTML element. Methods on `HTMLElement` include those that allow for the retrieval and modification of attributes that apply to all HTML elements.
- Specializations for all HTML elements that have attributes that extend beyond those specified in the `HTMLElement` interface. For all such attributes, the derived interface for the element contains explicit methods for setting and getting the values.
- The DOM Level 2 includes mechanisms to access and modify style specified through CSS and defines an event model that can be used with HTML documents.

NOTE

At the time of this writing, the DOM HTML Level 2 specification was at a working draft state, whereas all other portions of Level 2 (Core, Events, and so on) were complete and a full recommendation. For that reason, some of the descriptions and information in this chapter could change before the specification is final.

HTMLAnchorElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The `HTMLAnchorElement` object represents an HTML anchor element. The `HTMLAnchorElement` inherits all methods and properties from the `HTMLElement` object. Table 11.1 contains a list of properties and methods for this object.

Table 11.1 Contents of the `HTMLAnchorElement` Object

Type	Item	Description
Property	<code>accessKey</code>	Single character access key used to give access to this element
	<code>charset</code>	The character encoding of the linked resource
	<code>coords</code>	List of lengths defining active region
	<code>href</code>	URI of the linked resource
	<code>hrefLang</code>	The language code of the linked resource
	<code>name</code>	The name of this anchor
	<code>rel</code>	Forward link type
	<code>rev</code>	Reverse link type
	<code>shape</code>	The shape of the active area of this anchor element

Type	Item	Description
Method	tabIndex	Number indicating the tab order for this anchor element
	target	The frame to render the linked resource for this anchor element
	type	Advisory content type for this anchor element
	blur()	Removes keyboard focus from this anchor element
	focus()	Gives keyboard focus to this anchor element

Example

Listing 11.1 illustrates the creation of an `HTMLBodyElement` using the `HTMLDocument` object element and then setting its `alink` property.

Listing 11.1 *Creating an* `HTMLAnchorElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");

// -->
</script>
</html>
```

HTMLAnchorElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.accessKey`

Description

The `accessKey` property of the `HTMLAnchorElement` object is a single character access key to give access to the form control.

Example

Listing 11.2 illustrates the creation of an `HTMLAnchorElement` using the `HTMLDocument` object element and then setting its `accessKey` property.

Listing 11.2 *Creating an* `HTMLAnchorElement` *and Setting Its* `accessKey` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

Listing 11.2 Continued

```
var anchorObj = bodyObj.createElement("a");
anchorObj.accessKey = "A";
// -->
</script>
</html>
```

HTMLAnchorElement.blur()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlAnchorObj.blur()
```

Description

The `blur()` method of the `HTMLAnchorElement` object removes keyboard focus from this element.

Example

Listing 11.3 illustrates the creation of an `HTMLAnchorElement` using the `HTMLDocument` object `element` and then calling its `blur()` method.

Listing 11.3 Creating an `HTMLBodyElement` invoking its `blur()` method

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.blur();
// -->
</script>
</html>
```

HTMLAnchorElement.charset

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlAnchorObj.charset
```

Description

The `charset` property of the `HTMLAnchorElement` object represents the character encoding of the linked resource.

Example

Listing 11.4 illustrates the creation of an `HTMLAnchorElement` object using the `HTMLBodyElement` object and then setting its `charset` property.

Listing 11.4 *Creating an `HTMLAnchorElement` Object and Setting Its `charset` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.charset = "us/english";
// -->
</script>
</html>
```

HTMLAnchorElement.coords

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.coords`

Description

The `coords` property of the `HTMLAnchorElement` object is a comma-separated list of lengths, defining an active region's geometry. See also `shape` for the shape of the region.

Example

Listing 11.5 illustrates the creation of an `HTMLAnchorElement` object using the `HTMLBodyElement` object and then setting its `coords` property.

Listing 11.5 *Creating an `HTMLAnchorElement` and Setting Its `coords` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.coords = "12,34,10, 10";
// -->
</script>
</html>
```

HTMLAnchorElement.focus()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.focus()`

Description

The `focus()` method of the `HTMLAnchorElement` object gives keyboard focus to this element.

Example

Listing 11.6 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object and the invocation of its `focus()` method.

Listing 11.6 *Creating an HTMLAnchorElement and Invoking Its focus() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.focus();
// -->
</script>
</html>
```

HTMLAnchorElement.href

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.href`

Description

The `href` property of the `HTMLAnchorElement` object represents the URI of the linked resource.

Example

Listing 11.7 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object and then setting its `href` property.

Listing 11.7 *Creating an `HTMLAnchorElement` and Setting Its `href` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.href = "http://foo.bar/";
// -->
</script>
</html>
```

HTMLAnchorElement.hrefLang

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.hrefLang`

Description

The `hrefLang` property of the `HTMLAnchorElement` object represents the language code of the linked resource.

Example

Listing 11.8 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object element and then setting its `hrefLang` property.

Listing 11.8 *Creating an `HTMLAnchorElement`, and then Setting Its `hrefLang` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.hrefLang = "us/english";
// -->
</script>
</html>
```

HTMLAnchorElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.name`

Description

The `name` property of the `HTMLAnchorElement` object represents the anchor name.

Example

Listing 11.9 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object and then setting its `name` property.

Listing 11.9 *Creating an `HTMLAnchorElement` and Setting Its `name` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.name = "Link to foo site";
// -->
</script>
</html>
```

HTMLAnchorElement.rel

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.rel`

Description

The `rel` property of the `HTMLAnchorElement` object represents a forward link type.

Example

Listing 11.10 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object element and then setting its `rel` property.

Listing 11.10 *Creating an `HTMLAnchorElement` and Setting Its `name` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.rel = "http://foo.bar/link.htm";
// -->
</script>
</html>
```

HTMLAnchorElement.rev

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.rev`

Description

The rev property of the HTMLAnchorElement object represents a reverse link type.

Example

Listing 11.11 illustrates the creation of an HTMLAnchorElement using the HTMLBodyElement object and then setting its rev property.

Listing 11.11 *Creating an HTMLAnchorElement and Setting Its rev Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.rev = "http://foo.bar/link.htm";
// -->
</script>
</html>
```

HTMLAnchorElement.shape

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAnchorObj.shape

Description

The shape property of the HTMLAnchorElement object represents the shape of the active area. The coordinates are given by coords.

Example

Listing 11.12 illustrates the creation of an HTMLAnchorElement using the HTMLBody object and then setting its shape property.

Listing 11.12 *Creating an HTMLAnchorElement and Setting its shape Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.shape = "oval";
// -->
</script>
</html>
```

HTMLAnchorElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.tabIndex`

Description

The `tabIndex` property of the `HTMLAnchorElement` object represents the index that represents the element's position in the tabbing order.

Example

Listing 11.13 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object and then setting its `tabIndex` property.

Listing 11.13 *Creating an HTMLAnchorElement and Setting Its tabIndex Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.tabIndex = 3;
// -->
</script>
</html>
```

HTMLAnchorElement.target

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAnchorObj.target`

Description

The `target` property of the `HTMLAnchorElement` object represents the frame to render the resource in.

Example

Listing 11.14 illustrates the creation of an `HTMLAnchorElement` using the `HTMLBodyElement` object and then setting its `target` property.

Listing 11.14 *Creating an HTMLAnchorElement and Setting Its target Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.target = "frame1";
// -->
</script>
</html>
```

HTMLAnchorElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAnchorObj.type

Description

The type property of the HTMLAnchorElement object represents an advisory content type.

Example

Listing 11.15 illustrates the creation of an HTMLAnchorElement using the HTMLBodyElement object and then setting its type property.

Listing 11.15 *Creating an HTMLAnchorElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var anchorObj = bodyObj.createElement("a");
anchorObj.type = "text/html";
// -->
</script>
</html>
```

HTMLAppletElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML applet element. The `HTMLAppletElement` inherits all properties and methods of the `HTMLElement` object. Table 11.2 contains a list of properties and methods for this object.

Table 11.2 Contents of the `HTMLAppletElement` Object

Type	Item	Description
Property	<code>align</code>	The horizontal alignment for the applet element
	<code>alt</code>	Alternate text to display in place of the applet
	<code>archive</code>	List of archive files associated with this applet
	<code>code</code>	The file containing the applet
	<code>codeBase</code>	Base URI indicating the location of Code and Archive
	<code>height</code>	The height of the applet
	<code>hspace</code>	The horizontal space to the left and right of the applet
	<code>name</code>	The name of this applet
	<code>object</code>	The serialized applet file
	<code>vspace</code>	The space above and below the applet element
	<code>width</code>	The width of the applet

Example

Listing 11.16 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.16 Creating an `HTMLAppletElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
// -->
</script>
</html>
```

HTMLAppletElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.align`

Description

The `align` property of the `HTMLAppletElement` object aligns this object (vertically or horizontally) with respect to its surrounding text.

Example

Listing 11.17 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.17 *Creating an `HTMLAppletElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.align = "bottom";
// -->
</script>
</html>
```

HTMLAppletElement.alt

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.alt`

Description

The `alt` property of the `HTMLAppletElement` object represents the alternate text for user agents not rendering the normal content of this element.

Example

Listing 11.18 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its `alt` property.

Listing 11.18 *Creating an `HTMLAppletElement` and Setting Its `alt` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.alt = "Cannot display applet!";
// -->
</script>
</html>
```

HTMLAppletElement.archive

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.archive`

Description

The archive property of the HTMLAppletElement object is a comma-separated archive list.

Example

Listing 11.19 illustrates the creation of an HTMLAppletElement using the HTMLBodyObject object and then setting its archive property.

Listing 11.19 *Creating an HTMLAppletElement Object and Setting its archive Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.archive = "foo.jar,bar.jar";
// -->
</script>
</html>
```

HTMLAppletElement.code

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.code`

Description

The code property of the HTMLAppletElement object is the applet class file.

Example

Listing 11.20 illustrates the creation of an HTMLAppletElement using the HTMLBodyElement object element and then setting its code property.

Listing 11.20 *Creating an HTMLAppletElement and Setting Its code Property*

```
<html>
<script language="JavaScript" type="text/javascript">
```

```
<!--  
var appletObj = bodyObj.createElement("applet");  
appletObj.code = "foo.bar.applets.FooApplet";  
// -->  
</script>  
</html>
```

HTMLAppletElement.codeBase

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.codeBase`

Description

The codeBase property of the HTMLAppletElement object is an optional base URI for applet.

Example

Listing 11.21 illustrates the creation of an HTMLAppletElement using the HTMLBodyElement object and then setting its codeBase property.

Listing 11.21 *Creating an HTMLAppletElement Object and Setting Its code-base Property*

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var appletObj = bodyObj.createElement("applet");  
appletObj.codebase = "http://foo.bar/applets/";  
// -->  
</script>  
</html>
```

HTMLAppletElement.height

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.height`

Description

The height property of the HTMLAppletElement object allows you to override height.

Example

Listing 11.22 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its height property.

Listing 11.22 *Creating an `HTMLAppletElement` and Setting Its height Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.height = "25";
// -->
</script>
</html>
```

HTMLAppletElement.hspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.hspace`

Description

The `hspace` property of the `HTMLAppletElement` object represents the horizontal space to the left and right of this applet.

Example

Listing 11.23 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its `hspace` property.

Listing 11.23 *Creating an `HTMLAppletElement` and Setting Its hspace Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.hspace = 2;
// -->
</script>
</html>
```

HTMLAppletElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.name`

Description

The name property of the HTMLAppletElement object contains the name of the applet.

Example

Listing 11.24 illustrates the creation of an HTMLAppletElement using the HTMLBodyElement object and then setting its name property.

Listing 11.24 Creating an HTMLAppletElement and Setting Its name Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.name = "color selector applet";
// -->
</script>
</html>
```

HTMLAppletElement.object

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAppletObj.object`

Description

The object property of the HTMLAppletElement object represents the serialized applet file.

Example

Listing 11.25 illustrates the creation of an HTMLAppletElement using the HTMLBodyElement object and then setting its object property.

Listing 11.25 *Creating an HTMLAppletElement and Setting Its object Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.object = "http://foo.bar/applets/colorapplet.ser"
// -->
</script>
</html>
```

HTMLAppletElement.vspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAppletObj.vspace

Description

The *vspace* property of the *HTMLAppletElement* object represents the vertical space above and below the applet.

Example

Listing 11.26 illustrates the creation of an *HTMLAppletElement* using the *HTMLBodyElement* object and then setting its *vspace* property.

Listing 11.26 *Creating an HTMLAppletElement and Setting Its vspace Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.vspace = 5;
// -->
</script>
</html>
```

HTMLAppletElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAppletObj.width

Description

The *width* property of the *HTMLAppletElement* object allows you to override width.

Example

Listing 11.27 illustrates the creation of an `HTMLAppletElement` using the `HTMLBodyElement` object and then setting its width property.

Listing 11.27 *Creating an `HTMLAppletElement` and Setting Its width Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var appletObj = bodyObj.createElement("applet");
appletObj.width = 30;
// -->
</script>
</html>
```

HTMLAreaElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML area element. Table 11.3 contains a list of properties and methods for this object.

Table 11.3 *Contents of `HTMLAreaElement` Object*

Type	Item	Description
Property	<code>accessKey</code>	Key that gives access to this area element
	<code>alt</code>	Alternate text to text to be used in place of this area element
	<code>coords</code>	List of lengths defining active area on this area element
	<code>href</code>	URI of the linked resource
	<code>noHref</code>	Specified that this area isn't active if true
	<code>shape</code>	The shape of the active area
	<code>tabIndex</code>	Number representing the tab order for this area element
	<code>target</code>	Frame to render the lined resource in

Example

Listing 11.28 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object and then setting its `accessKey` property.

Listing 11.28 *Creating an* `HTMLAreaElement` *Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
// -->
</script>
</html>
```

HTMLAreaElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.accessKey`

Description

The `accessKey` property of the `HTMLAreaElement` object is a single character access key to give access to the form control.

Example

Listing 11.29 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object and then setting its `accessKey` property.

Listing 11.29 *Creating an* `HTMLAreaElement` *and Setting Its* `accessKey` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.AccessKey = "Z";// -->
</script>
</html>
```

HTMLAreaElement.alt

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.alt`

Description

The `alt` property of the `HTMLAreaElement` object represents the alternate text for user agents not rendering the normal content of this element.

Example

Listing 11.30 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object element and then setting its `alt` property.

Listing 11.30 Creating an `HTMLAreaElement` and Setting Its `alt` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.alt = "here is area 1";

// -->
</script>
</html>
```

HTMLAreaElement.coords

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.coords`

Description

The `coords` property of the `HTMLAreaElement` object is a comma-separated list of lengths, defining an active region's geometry.

Example

Listing 11.31 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object and then setting its `coords` property.

Listing 11.31 Creating an `HTMLAreaElement` and Setting Its `coords` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.coords = "23,45,34,44";

// -->
</script>
</html>
```

HTMLAreaElement.href

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.href`

Description

The href property of the HTMLAreaElement object represents the URI of the linked resource.

Example

Listing 11.32 illustrates the creation of an HTMLAreaElement using the HTMLMapElement object and then setting its href property.

Listing 11.32 *Creating an HTMLAreaElement and Setting Its href Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.href = "http://foo.bar/link.html";

// -->
</script>
</html>
```

HTMLAreaElement.noHref

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.noHref`

Description

The noHref property of the HTMLAreaElement object specifies that this area is inactive; that is, it has no associated action.

Example

Listing 11.33 illustrates the creation of an HTMLAreaElement using the HTMLMapElement object and then setting its noHref property.

Listing 11.33 *Creating an HTMLAreaElement and Setting Its noHref Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.noHref = true;

// -->
</script>
</html>
```

HTMLAreaElement.shape

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAreaObj.shape

Description

The shape property of the HTMLAreaElement object represents the shape of the active area. The coordinates are given by coordinates.

Example

Listing 11.34 illustrates the creation of an HTMLAreaElement using the HTMLMapElement object and then setting its shape property.

Listing 11.34 *Creating an HTMLAreaElement and Setting Its shape Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.ccreateElement("area");
areaObj.shape = "rect";

// -->
</script>
</html>
```

HTMLAreaElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlAreaObj.tabIndex

Description

The `tabIndex` property of the `HTMLAreaElement` object is a index that represents the element's position in the tabbing order.

Example

Listing 11.35 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object and then setting its `tabIndex` property.

Listing 11.35 Creating an HTMLAreaElement and Setting Its tabIndex Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.TabIndex = 3;

// -->
</script>
</html>
```

HTMLAreaElement.target

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlAreaObj.target`

Description

The `target` property of the `HTMLAreaElement` object represents the frame to render the resource in.

Example

Listing 11.36 illustrates the creation of an `HTMLAreaElement` using the `HTMLMapElement` object and then setting its `target` property.

Listing 11.36 Creating an HTMLAreaElement and Setting Its target Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaObj = mapObj.createElement("area");
areaObj.target = "mainframe";
// -->
```

```
</script>  
</html>
```

HTMLBaseElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The HTMLBaseElement object encapsulates the Document base URI. HTMLBaseElement inherits all methods and properties from HTMLElement. Table 11.4 contains a list of properties and methods for this object.

Table 11.4 *Properties of the* HTMLBaseElement *Object*

Item	Description
href	The base URI of the document
target	Default target frame

Example

Listing 11.37 illustrates the creation of an HTMLBaseElement using the HTMLDocument object.

Listing 11.37 *Creating an* HTMLBaseElement

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var baseElement = htmlDoc.createElement("base");  
// -->  
</script>  
</html>
```

HTMLBaseElement.href

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlBaseElemObj.href

Description

The href property of the HTMLBaseElement object represents the base URI for this element.

Example

Listing 11.38 illustrates the creation of an HTMLBaseElement using the HTMLDocument object element and then setting its href and target properties.

Listing 11.38 Creating an HTMLBaseElement and Setting Its href Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var baseElement = htmlDoc.createElement("base");
baseElement.href = "http://foo.bar/";

// -->
</script>
</html>
```

HTMLBaseElement.target

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlBaseElemObj.target

Description

The target property of the HTMLBaseElement object represents the default target frame.

Example

Listing 11.39 illustrates the creation of an HTMLBaseElement using the HTMLDocument object element and then setting its target property.

Listing 11.39 Creating an HTMLBaseElement and Setting Its target Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var baseElement = htmlDoc.createElement("base");
```

```
baseElement.target = "MainFrame";  
// -->  
</script>  
</html>
```

HTMLBaseFontElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML BASEFONT element. The HTMLBaseFontElement inherits all methods and properties from the HTMLElement object. Table 11.5 contains a list of properties for this object.

Table 11.5 *Properties of the* HTMLBaseFontElement *object*

Item	Description
color	The color of this base font element
face	The font face identifier for this base font element
size	The font size for this base font element

Example

Listing 11.40 illustrates the creation of an HTMLBaseFontElement using the HTMLBodyElement object.

Listing 11.40 *Creating an* HTMLBaseFontElement *Object*

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var baseFontObj = bodyObj.createElement("basefont");  
  
// -->  
</script>  
</html>
```

HTMLBaseFontElement.color

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlBaseFontObj.color

Description

The color property of the HTMLBaseFontElement object represents the color of this font.

Example

Listing 11.41 illustrates the creation of an HTMLBaseFontElement using the HTMLBody object and then setting its color property.

Listing 11.41 Creating an HTMLBaseFontElement and Setting Its color Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var baseFontObj = bodyObj.createElement("basefont");
baseFontObj.color = "#FFFFFF";
// -->
</script>
</html>
```

HTMLBaseFontElement.face

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlBaseFontObj.face

Description

The face property of the HTMLBaseFontElement object is the font face identifier.

Example

Listing 11.42 illustrates the creation of an HTMLBaseFontElement using the HTMLBodyElement object and then setting its face property.

Listing 11.42 Creating an HTMLBaseFontElement and Setting Its face Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var baseFontObj = bodyObj.createElement("basefont");
baseFontObj.face = "courier";
// -->
</script>
</html>
```

HTMLBaseFontElement.size

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBaseFontObj.size`

Description

The size property of the HTMLBaseFontElement object is the font size.

Example

Listing 11.43 illustrates the creation of an HTMLBaseFontElement using the HTMLBodyElement object and then setting its size property.

Listing 11.43 *Creating an HTMLBaseFontElement and Setting Its size Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var baseFontObj = bodyObj.createElement("basefont");
baseFontObj.size = 12;
// -->
</script>
</html>
```

HTMLBodyElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object encapsulates the HTML document body. This element is always present in the DOM API, even if the tags are not present in the source document. HTMLBodyElement inherits all methods and properties from HTMLElement. Table 11.6 contains a list of properties for this object.

Table 11.6 *Properties of the HTMLBodyElement Object*

Item	Description
aLink	Color of active links in the body element
background	URI of background texture image tile
bgColor	The background color for this document
link	Color of links that are not visited and active

Table 11.6 *Continued*

Item	Description
text	Text color for this document
vLink	Color for visited links

Example

Listing 11.44 illustrates the creation of an HTMLBodyElement.

Listing 11.44 *Creating an HTMLBodyElement Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");

// -->
</script>
</html>
```

HTMLBodyElement.aLink

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlBodyObj.aLink
```

Description

The aLink property of the HTMLBodyElement object represents the color of active links (after mouse button down, but before mouse button up).

Example

Listing 11.45 illustrates the creation of an HTMLBodyElement using the HTMLDocument object element and then setting its aLink property.

Listing 11.45 *Creating an HTMLBodyElement and Setting Its aLink Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
bodyObj.aLink = "#FF0000"
```

```
// -->  
</script>  
</html>
```

HTMLBodyElement.background

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBodyObj.background`

Description

The background property of the HTMLBodyElement object is the URI of the background texture tile image.

Example

Listing 11.46 illustrates the creation of an HTMLBodyElement using the HTMLDocument object and then setting its background property.

Listing 11.46 *Creating an HTMLBodyElement and Setting Its background Property*

```
<html>  
<script language="JavaScript" type="text/javascript">  
  <!--  
  var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
  htmlDoc.open();  
  var bodyObj = htmlDoc.createElement("body");  
  bodyObj.background = "http://foo.bar/images/tile.jpg"  
  // -->  
</script>  
</html>
```

HTMLBodyElement.bgColor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBodyObj.bgColor`

Description

The bgColor property of the HTMLBodyElement object is the document background color.

Example

Listing 11.47 illustrates the creation of an `HTMLBodyElement` using the `HTMLDocument` object element and then setting its `bgColor` property.

Listing 11.47 Creating an `HTMLBodyElement` and Setting Its `bgColor` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
bodyObj.bgColor = "#FFFFFF";
// -->
</script>
</html>
```

HTMLBodyElement.link

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBodyObj.link`

Description

The `link` property of the `HTMLBodyElement` object is the color of links that are inactive and unvisited.

Example

Listing 11.48 illustrates the creation of an `HTMLBodyElement` using the `HTMLDocument` object element and then setting its `link` property.

Listing 11.48 Creating an `HTMLBodyElement` and Setting Its `link` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
bodyObj.link = "#000066"
// -->
</script>
</html>
```

HTMLBodyElement.text

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBodyObj.text`

Description

The text property of the HTMLBodyElement object is the document text color.

Example

Listing 11.49 illustrates the creation of an HTMLBodyElement using the HTMLDocument object element and then setting its text property.

Listing 11.49 *Creating an HTMLBodyElement and Setting Its text Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
bodyObj.text = "#000000"
// -->
</script>
</html>
```

HTMLBodyElement.vLink

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlBodyObj.vLink`

Description

The vLink property of the HTMLBodyElement object represents the color of links that have been visited by the user.

Example

Listing 11.50 illustrates the creation of an HTMLBodyElement using the HTMLDocument object element and then setting its vLink property.

Listing 11.50 *Creating an HTMLBodyElement and Setting Its vLink Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
bodyObj.vLink = "#6666CC"
// -->
</script>
</html>
```

HTMLBRElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents the HTML element that forces a line break. The HTMLBRElement inherits all methods and properties of the HTMLElement object. The property for this object is as follows:

<i><u>Item</u></i>	<i><u>Description</u></i>
clear	Controls flow of text

Example

Listing 11.51 illustrates the creation of an HTMLBRElement using the HTMLBodyElement.

Listing 11.51 *Creating an HTMLBRElement Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var brObj = bodyObj.createElement("br");

// -->
</script>
</html>
```

HTMLBRElement.clear

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlBRObj.clear

Description

The `clear` property of the `HTMLBRElement` object controls flow of text around floats.

Example

Listing 11.52 illustrates the creation of an `HTMLBRElement` using the `HTMLBodyElement` object and then setting its `clear` property.

Listing 11.52 *Creating an `HTMLBRElement` and Setting Its `clear` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var brObj = bodyObj.createElement("br");
brObj.clear = "left";
// -->
</script>
</html>
```

HTMLButtonElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML button element. It inherits all methods and properties of the `HTMLElement` object. Table 11.7 contains a list of properties for this object.

Table 11.7 *Properties of the `HTMLButtonElement` Object*

Item	Description
<code>accessKey</code>	A character access key that gives control to this button element
<code>disabled</code>	Indicates whether this button element is disabled
<code>form</code>	The form that this button element resides in
<code>name</code>	The name of this button element
<code>tabIndex</code>	Number that represents the tab order for this button element
<code>type</code>	The type of this button element
<code>value</code>	The value of this button element

Example

Listing 11.53 illustrates the creation of a `HTMLButtonElement` using a `HTMLFormElement` and setting its `type` property.

Listing 11.53 *Creating an HTMLButtonElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var buttonObj = formObj.createElement("button");
buttonObj.type = "submit";
// -->
</script>
</html>
```

HTMLButtonElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlButtonObj.accessKey

Description

The `accessKey` property of the `HTMLButtonElement` object is a single character access key to give access to the form control.

Example

Listing 11.54 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object and then setting its `accessKey` property.

Listing 11.54 *Creating an HTMLButtonElement and Setting Its accessKey Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.accessKey = "B";
// -->
</html>
```

HTMLButtonElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlButtonObj.disabled

Description

The `disabled` property of the `HTMLButtonElement` object allows you to disable button instances.

Example

Listing 11.55 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object and then setting its `disabled` property.

Listing 11.55 *Creating an `HTMLButtonElement` and Setting Its disabled Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.disabled = true;
// -->
</script>
</html>
```

HTMLButtonElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlButtonObj.form`

Description

The `form` property of the `HTMLButtonElement` object returns the FORM element containing this control. It returns `Null` if this control isn't within the context of a form.

Example

Listing 11.56 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object and then reading its `form` Property.

Listing 11.56 *Creating an `HTMLButtonElement` and Reading its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
if(buttonObj.form.name == "emailForm")
    processEmail(buttonObj.form);
// -->
</script>
</html>
```

HTMLButtonElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlButtonObj.name`

Description

The `name` property of the `HTMLButtonElement` object represents a form control or object name when submitted with a form.

Example

Listing 11.57 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object and then setting its `name` property.

Listing 11.57 *Creating an HTMLButtonElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.name = "foo";
// -->
</script>
</html>
```

HTMLButtonElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlButtonObj.tabIndex`

Description

The `tabIndex` property of the `HTMLButtonElement` object is the index that represents the element's position in the tabbing order.

Example

Listing 11.58 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object and then setting its `tabIndex` property.

Listing 11.58 *Creating an HTMLButtonElement and Setting Its tabIndex Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.tabIndex = 10;
// -->
</script>
</html>
```

HTMLButtonElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlButtonObj.type`

Description

The type property of the HTMLButtonElement object represents the type of button.

Example

Listing 11.59 illustrates the creation of an HTMLButtonElement using the HTMLFormElement object element and then setting its type property.

Listing 11.59 *Creating an HTMLButtonElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.type = "submit";
// -->
</script>
</html>
```

HTMLButtonElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlButtonObj.value`

Description

The value property of the HTMLButtonElement object is the current form control value.

Example

Listing 11.60 illustrates the creation of an `HTMLButtonElement` using the `HTMLFormElement` object element and then setting its `value` property.

Listing 11.60 *Creating an `HTMLButtonElement` and Setting Its value Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var buttonObj = formObj.createElement("button");
buttonObj.type = "submit";
buttonObj.value = "Submit";
// -->
</script>
</html>
```

HTMLCollection

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

An `HTMLCollection` is a list of nodes. An individual node can be accessed by either ordinal index or the node's name or ID attributes.



NOTE

Collections in the HTML DOM are assumed to be *live* meaning that they are automatically updated when the underlying document is changed.

Table 11.8 contains a list of the property and methods for this object.

Table 11.8 *Contents of the `HTMLCollection` Object*

Type	Item	Description
Property	<code>length</code>	The number of items in this collection
Method	<code>item()</code>	Returns the item at a certain index
	<code>namedItem()</code>	Returns a member of the collection whose name is specified

Example

Listing 11.61 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object then iterating through that document's images property processing each image.

Listing 11.61 Iterating Through an HTMLCollection

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocImages = htmlDoc.images;
var i = 0;
while(i < htmlDocImages.length) {
    var image = htmlDocImages.item(i);
    processImageSrc(imageSrc);
    i++;
}
</script>
</html>
```

HTMLCollection.item()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlCollectionObj.item(index)

Description

The `item()` method of the `HTMLCollection` object retrieves a node specified by number *index*. Nodes are numbered in tree order (depth-first traversal order).

Example

Listing 11.63 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then iterating through that document's `images` property processing each image.

Listing 11.63 Iterating Through an HTMLCollection

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocImages = htmlDoc.images;
var i = 0;
while(i < htmlDocImages.length) {
    var image = htmlDocImages.item(i);
    processImageSrc(imageSrc);
    i++;
}
```

Listing 11.61 Continued

```
}  
</script>  
</html>
```

HTMLCollection.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlCollectionObj.length

Description

The length property of the HTMLCollection object specifies the length or size of the list.

Example

Listing 11.62 illustrates the creation of an HTMLDocument using the HTMLDOMImplementation object and then iterating through that document's images property processing each image.

Listing 11.62 Iterating Through an HTMLCollection *Using the* length Property

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var htmlDocImages = htmlDoc.images;  
var i = 0;  
while(i < htmlDocImages.length) {  
    var image = htmlDocImages.item(i);  
    processImageSrc(imageSrc);  
    i++;  
}  
</script>  
</html>
```

HTMLCollection.namedItem()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlCollectionObj.namedItem(*name*)

Description

The `namedItem()` method of the `HTMLCollection` object retrieves a `Node` using a *name*. It first searches for a `Node` with a matching `ID` attribute. If it doesn't find one, it then searches for a `Node` with a matching `name` attribute, but only on those elements that are allowed a `name` attribute.

Example

Listing 11.64 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and the iteration through that document's `images` property processing each image.

Listing 11.64 *Iterating Through an `HTMLCollection` Using the `namedItem()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocImages = htmlDoc.images;
var i = 0;
while(i < htmlDocImages.Length) {
    var image = htmlDocImages.namedItem("iconImage");
    processImageSrc(imageSrc);
    i++;
}
</script>
</html>
```

HTMLDirectoryElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML directory element. The `HTMLDirectoryElement` object inherits all methods and properties from `HTMLElement`. The property for this object is as follows:

<i><u>Item</u></i>	<i><u>Description</u></i>
<code>compact</code>	Reduces the spacing between list items when true

Example

Listing 11.65 illustrates the creation of an `HTMLDirectoryElement` using the `HTMLDListElement`.

Listing 11.65 *Creating an* HTMLDirectory

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var dListObj = bodyObj.createElement("dlist");
var directoryObj = dListObj.createElement("directory");

// -->
</script>
</html>
```

HTMLDirectoryElement.compact

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDirectoryObj.compact

Description

The compact property of the HTMLDirectoryElement object reduces the spacing between list items.

Example

Listing 11.66 illustrates the creation of an HTMLDirectoryElement using the HTMLDListElement object and then setting its compact property.

Listing 11.66 *Creating an* HTMLDirectoryElement *and Setting Its* compact Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var dListObj = bodyObj.createElement("dlist");
var directoryObj = dListObj.createElement("directory");
directoryObj.compact = true;
// -->
</script>
</html>
```

HTMLDivElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML generic block container. The HTMLDivElement object inherits all methods and properties from HTMLElement. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
align	Horizontal text alignment for this DIV element

Example

Listing 11.67 illustrates the creation of an HTMLDivElement using the HTMLBodyElement object and then setting its align property.

Listing 11.67 Creating an HTMLDivElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var divObj = bodyObj.createElement("div");
divObj.align = "top";
// -->
</script>
</html>
```

HTMLDivElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDivObj.align

Description

The align property of the HTMLDivElement object represents the horizontal text alignment.

Example

Listing 11.68 illustrates the creation of an HTMLDivElement using the HTMLBodyElement object and then setting its align property.

Listing 11.68 Creating an HTMLDivElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var divObj = bodyObj.createElement("div");
divObj.align = "bottom";
// -->
</script>
</html>
```

HTMLDListElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML directory list element. The HTMLDListElement object inherits all methods and properties from HTMLElement. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
compact	Reduces the spacing between list items if true

Example

Listing 11.69 illustrates the creation of an HTMLDListElement using the HTMLBodyElement object element and then setting its compact property.

Listing 11.69 *Creating an HTMLDListElement and Setting Its compact Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var dListObj = bodyObj.createElement("dlist");
dListObj.compact = true;
// -->
</script>
</html>
```

HTMLDListElement.compact

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDListObj.compact

Description

The compact property of the HTMLDListElement object reduces the spacing between list items.

Example

Listing 11.70 illustrates the creation of an HTMLDListElement using the HTMLBodyElement object element and then setting its compact property.

Listing 11.70 *Creating an HTMLDListElement and Setting Its compact Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var dListObj = bodyObj.createElement("dlist");
dListObj.compact = true;
// -->
</script>
</html>
```

HTMLDocument

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

An HTMLDocument is the root of the HTML hierarchy and holds the entire content. Besides providing access to the hierarchy, it also provides some convenience methods for accessing certain sets of information from the document. Table 11.9 contains a list of properties and methods for this object.

Table 11.9 *Contents of the HTMLDocument Object*

Type	Item	Description
Property	anchors	A list of anchors present in this document
	applets	A list of applets displayed on this document
	cookie	A cookie associated with this document
	body	The text representing the body of this document
	domain	The domain that this document was delivered from
	forms	A list of forms present in this document
	images	A list of images displayed in this document
	links	A list of external links in this document
	referrer	The referring URL to this document
	title	The title of this HTML document
	url	The fully qualified URL where this document is located
Method	close()	Closes a document stream
	getElementsByName()	Returns a NodeList of all elements with the specified name

Table 11.9 Continued

Type	Item	Description
	open()	Opens a document stream for writing
	write()	Writes to an opened document stream
	writeln()	Writes to an opened document stream

Example

Listing 11.71 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation`.

Listing 11.71 Creating a New `HTMLDocument`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textArea = htmlDoc.createElement("textarea");
textArea.value = "Here's some text in a TEXTAREA");
// -->
</script>
</html>
```

HTMLDocument.anchors

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlDoc.anchors
```

Description

The `anchors` property of the `HTMLDocument` object is a collection of all the anchor (A) elements in a document with a value for the `name` attribute.

NOTE

For reasons of backward compatibility, the returned set of anchors only contains those anchors created with the `name` attribute, not those created with the `ID` attribute.

Example

Listing 11.72 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object then iterating through that document's anchors property processing each anchor element.

Listing 11.72 Iterating Through a Document's anchors

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocAnchors = htmlDoc.anchors;
var i = 0;
while(i < htmlDocAnchors.length) {
    var anchor = htmlDocAnchors.item(i);
    processAnchor(anchor);
    i++;
}
// -->
</script>
</html>
```

HTMLDocument.applets

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDoc.applets

Description

The applets property of the HTMLDocument object is a collection of all the OBJECT elements that include applets and APPLET (deprecated) elements in a document.

Example

Listing 11.73 illustrates the creation of an HTMLDocument using the HTMLDOMImplementation object then iterating through that document's applets property processing each applet.

Listing 11.73 Iterating Through a Document's applets

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocApplets = htmlDoc.applets;
var i = 0;
while(i < htmlDocApplets.length) {
    var applet = htmlDocApplets.item(i);
    processImageSrc(applet);
    i++;
}
}
```

Listing 11.73 Continued

```
// -->
</script>
</html>
```

HTMLDocument.body

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.body`

Description

The `body` property of the `HTMLDocument` object is the element that contains the content for the document. In documents with `BODY` contents, this returns the `BODY` element. In frameset documents, this returns the outermost `FRAMESET` element.

Example

Listing 11.74 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation` and then changes the background color of the `body` property.

Listing 11.74 Inspecting and Modifying the `body` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var docBody = htmlDoc.body;
docBody.bgColor = "#FFFFFF";
// -->
</script>
</html>
```

HTMLDocument.close()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.close()`

Description

The `close()` method of the `HTMLDocument` object closes a document stream opened by `open()` and forces rendering.

Example

Listing 11.75 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then using the `open()` method to open this document for writing. A cookie is added and then the document is closed using `close()`.

Listing 11.75 *Closing an `HTMLDocument` Using the `close()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.cookie = "sessionId=3213;expires=34283028"
htmlDoc.close();
// -->
</script>
</html>
```

HTMLDocument.cookie

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.cookie`

Description

The `cookie` property of the `HTMLDocument` object represents the cookies associated with this document. If there are none, the value is an empty string. Otherwise, the value is a string: a semicolon-delimited list of `name=value` pairs for all the cookies associated with the page; for example, `name=value;expires=date`.

Example

Listing 11.76 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then accessing the `cookie` property and setting its value.

Listing 11.76 *Accessing the `cookie` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.cookie = "sessionId=3213;expires=34283028"
}
</html>
```

HTMLDocument.domain

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.domain`

Description

The domain property of the HTMLDocument object represents the domain name of the server that served the document, or Null if the server cannot be identified by a domain name.

Example

Listing 11.77 creates a new HTMLDocument object using the createHTMLDocument() method of HTMLDOMImplementation and then inspects the domain property to see which domain the page originated from.

Listing 11.77 *Inspecting the domain Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
if(htmlDoc.domain == "foo.bar")
    cameFromFooBar = true;
// -->
</script>
</html>
```

HTMLDocument.forms

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.forms`

Description

The forms property of the HTMLDocument object is a collection of all the forms of a document.

Example

Listing 11.78 illustrates the creation of an HTMLDocument using the HTMLDOMImplementation object and then iterating through that document's forms property processing the form with the name "emailForm".

Listing 11.78 Iterating Through a Document's forms

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocForms = htmlDoc.forms;
var i = 0;
while(i < htmlDocForms.length) {
    var form = htmlDocForms.Item(i);
    if(form.name == "emailForm")
        processEmailForm(form);
    i++;
}
// -->
</script>
</html>
```

HTMLDocument.getElementsByName()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDoc.getElementsByName(*name*)

Description

The `getElementsByName()` method of the `HTMLDocument` object returns the (possibly empty) collection of elements whose name value is given by *name*.

Example

Listing 11.79 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then iterating through that document's list of images using the `getElementsByName()` method processing each image element.

Listing 11.79 Iterating Through a Document's Images Using the getElementsByName() Method

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var imageList = htmlDoc.getElementsByName("image");
var i = 0;
```

Listing 11.79 Continued

```
while(i < imageList.length) {  
    var image = imageList.item(i);  
    processImage(image);  
    i++;  
}  
</html>
```

HTMLDocument.images

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDoc.images

Description

The `images` property of the `HTMLDocument` object is a collection of all the `IMG` elements in a document. The behavior is limited to `IMG` elements for backward compatibility.

Example

Listing 11.80 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then iterating through that document's `images` property processing each image.

Listing 11.80 Iterating Through an HTMLCollection

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var htmlDocImages = htmlDoc.images;  
var i = 0;  
while(i < htmlDocImages.length) {  
    var image = htmlDocImages.namedItem("iconImage");  
    processImageSrc(imageSrc);  
    i++;  
}  
// -->  
</script>  
</html>
```

HTMLDocument.links

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.links`

Description

The `links` property of the `HTMLDocument` object is a collection of all AREA elements and anchor (A) elements in a document with a value for the `href` attribute.

Example

Listing 11.81 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then iterating through that document's `links` property processing each applet.

Listing 11.81 Iterating Through a Document's links

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var htmlDocLinks = htmlDoc.links;
var i = 0;
while(i < htmlDocApplets.length) {
    var link = htmlDocLinks.item(i);
    processLink(link);
    i++;
}
// -->
</script>
</html>
```

HTMLDocument.open()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.open()`

Description

The `open()` method of the `HTMLDocument` object is used to open a document stream for writing. If a document exists in the target, this method clears it.

Example

Listing 11.82 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then using the `open()` method to open this document for writing.

Listing 11.82 *Opening an `HTMLDocument` Using the `open()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.cookie = "sessionId=3213;expires=34283028"
htmlDoc.close();
// -->
</script>
</html>
```

HTMLDocument.referrer

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.referrer`

Description

The `referrer` property of the `HTMLDocument` object returns the URI of the page that linked to this page. The value is an empty string if the user navigated to the page directly (not through a link, but, for example, via a bookmark).

Example

Listing 11.83 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation` and then inspects the `referrer` property to see which server the page originated from.

Listing 11.83 *Inspecting the `referrer` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
if(htmlDoc.referrer == "http://www.foo.bar/index.htm")
    cameFromFooBar = true;
// -->
</script>
</html>
```

HTMLDocument.title

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.title`

Description

The `title` property of the `HTMLDocument` object represents the title of a document as specified by the `TITLE` element in the head of the document.

Example

Listing 11.84 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation` and then changes the title.

Listing 11.84 Changing the title Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.title = "New Title for My HTML Doc";
// -->
</script>
</html>
```

HTMLDocument.URL

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlDoc.URL`

Description

The `URL` property of the `HTMLDocument` object represents the complete URI of the document.

Example

Listing 11.85 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation` and then creates a `TEXTAREA` element and sets its value to be that of the `URL` property.

Listing 11.85 *Inspecting the URL Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textArea = htmlDoc.createElement("textarea");
textArea.value = htmlDoc.URL;
// -->
</script>
</html>
```

HTMLDocument.write()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlDoc.write(string)
```

Description

The `write()` method of the `HTMLDocument` object is used to write a *string* to a document stream opened by `open()`. The *string* is parsed into the document's structure model.

Example

Listing 11.86 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then using the `write()` method to write text data to this document.

Listing 11.86 *Using the write() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.write("<h1>Hello World</h1>");
htmlDoc.close();

// -->
</html>
```

HTMLDocument.writeln()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlDoc.writeln(*string*)

Description

The `writeln()` method of the `HTMLDocument` object is used to write a *string* followed by a newline character to a document stream opened by `open()`. The *string* is parsed into the document's structure model.

Example

Listing 11.87 illustrates the creation of an `HTMLDocument` using the `HTMLDOMImplementation` object and then using the `writeln()` method to write text data to this document.

Listing 11.87 Using the `writeln()` Method

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
htmlDoc.writeln("<h1>Hello World</h1>");
htmlDoc.close();
// -->
</script>
</html>
```

HTMLDOMImplementation

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The `HTMLDOMImplementation` object inherits all properties and methods of the `DOMImplementation` object with a method for creating an HTML document instance. The method for this object is as follows:

<i>Item</i>	<i>Description</i>
<code>createHTMLDocument()</code>	Creates an HTML Document and returns it

Example

Listing 11.88 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation`.

Listing 11.88 *Creating a `HTMLDOMImplementation` object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textArea = htmlDoc.createElement("textarea");
textArea.Value = "Here's some text in a TEXTAREA");
// -->
</script>
</html>
```

HTMLDOMImplementation.createHTMLDocument()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlDOMImpl.createHTMLDocument(documentTitle);
```

Description

The `createHTMLDocument()` method of the `HTMLDOMImplementation` object creates an `HTMLDocument` object with the minimal tree made of the following elements: `HTML`, `HEAD`, `TITLE`, and `BODY`.

The `documentTitle` parameter names the title of the document to be set as the content of the `TITLE` element, through a child `Text` node.

Example

Listing 11.89 creates a new `HTMLDocument` object using the `createHTMLDocument()` method of `HTMLDOMImplementation`.

Listing 11.89 *Creating an `HTMLDocument` Using the `createHTMLDocument()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textArea = htmlDoc.createElement("textarea");
textArea.Value = "Here's some text in a TEXTAREA");
// -->
</script>
</html>
```


HTMLElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

All HTML element objects inherit from this object. The base HTMLElement object represents elements that only expose the HTML core attributes.

These elements are as follows:

- HEAD
- special: SUB, SUP, SPAN, BDO
- font: TT, I, B, U, S, STRIKE, BIG, SMALL
- phrase: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ACRONYM, ABBR
- list: DD, DT
- NOFRAMES, NOSCRIPT
- ADDRESS, CENTER

Table 11.10 contains a list of properties for this object.

Table 11.10 *Properties of the* HTMLElement *Object*

Item	Description
className	Class attribute of this element
dir	Specifies the base direction of this element's text
id	Element identifier for this element
lang	Language code for this element
title	This element's advisory title

Example

Listing 11.90 illustrates the creation of a BaseElement object interrogating its id to verify the correct type.

Listing 11.90 *Creation of a* BaseElement *Object and Reading Its* id *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
if(textAreaObj.id != "textarea")
    throw new DOMException("implementation error!");
```

Listing 11.90 Continued

```
// -->
</script>
</html>
```

HTMLElement.className

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlElemObj.className`

Description

The `className` property of the `HTMLElement` object is the class attribute of the element.

Example

Listing 11.91 illustrates the creation of a `TextAreaElement` object and setting its `className` property.

Listing 11.91 Creating a `TextAreaElement` Object and Setting Its `className` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
textAreaObj.className = "TextArea";
// -->
</html>
```

HTMLElement.dir

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlElemObj.dir`

Description

The `dir` property of the `HTMLElement` object specifies the base direction of directionally neutral text and the directionality of tables.

Example

Listing 11.92 illustrates the setting the `dir` property of the `HTMLElement` object.

Listing 11.92 *Setting the `dir` property of `HTMLElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
textAreaObj.dir = "right";
// -->
</html>
```

HTMLElement.id

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlElemObj.id

Description

The `id` property of the `HTMLElement` object represents the element's identifier.

Example

Listing 11.93 illustrates the creation of a `BaseElement` object interrogating its `ID` to verify the correct type.

Listing 11.93 *Reading the `id` property of `HTMLElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
if(textAreaObj.id != "textarea")
    throw new DOMException("implementation error!");
// -->
</html>
```

HTMLElement.lang

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlElemObj.lang

Description

The lang property of the HTML`Element` object represents the language code defined in RFC 1766, which can be found at <http://www.ietf.org>.

Example

Listing 11.94 illustrates writing if the lang property of the HTML`Element` object.

Listing 11.94 Writing the lang Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
textAreaObj.lang = "us/english";
// -->
</html>
```

HTML`Element`.title

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlElemObj.title`

Description

The title property of the HTML`Element` object is the element's advisory title.

Example

Listing 11.95 illustrates the creation of a HTML`Element` and setting its title.

Listing 11.95 Setting the title Property of the HTML`Element` Object

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var textAreaObj = htmlDoc.createElement("textarea");
textAreaObj.title = "TextArea Element";
// -->
</html>
```

HTMLFieldSetElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The `HTMLFieldSetElement` organizes form controls into logical groups. This object inherits all properties and methods from the `HTMLElement` object. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
<code>form</code>	The form in which this field set element resides

Example

Listing 11.96 illustrates the creation of `HTMLFieldSetElement` object

Listing 11.96 *Creating an HTMLFieldSetElement Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fieldSetObj = formObj.createElement("fieldset");
// -->
</script>
</html>
```

HTMLFieldSetElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormSetObj.form`

Description

The `form` property of the `HTMLFieldSetElement` returns the `FORM` element containing this control. Returns `Null` if this control isn't within the context of a form.

Example

Listing 11.97 illustrates the creation of an `HTMLFieldSetElement` using the `HTMLFormElement` object and then checking its `form` property.

Listing 11.97 *Creating an HTMLFieldSetElement and Getting Its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fieldSetObj = formObj.createElement("fieldset");
if(fieldSetObj.form.name == "emailForm")
    processEmail(fieldSetObj.Form);
// -->
</html>
```

HTMLFontElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML Font element. HTMLFontElement inherits all methods and properties of HTMLElement. Table 11.11 contains a list of properties for this object.

Table 11.11 *Properties of the HTMLFontElement Object*

Item	Description
color	The color for this font
face	The font face identifier for this font element
size	The size of this font element

Example

Listing 11.98 illustrates the creation of an HTMLFontElement using the HTMLBodyElement object.

Listing 11.98 *Creating an HTMLFontElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fontObj = bodyObj.createElement("font");

// -->
</script>
</html>
```

HTMLFontElement.color

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFontObj.color`

Description

The `color` property of the `HTMLFontElement` object represents the font color.

Example

Listing 11.99 illustrates the creation of an `HTMLFontElement` using the `HTMLBodyElement` object and then setting its `color` property.

Listing 11.99 *Creating an HTMLBodyElement and Setting Its color Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fontObj = bodyObj.createElement("font");
fontObj.color = "#FFFFFF";
// -->
</html>
```

HTMLFontElement.face

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFontObj.face`

Description

The `face` property of the `HTMLFontElement` object represents the font face identifier.

Example

Listing 11.100 illustrates the creation of an `HTMLFontElement` using the `HTMLBodyElement` object and then setting its `face` property.

Listing 11.100 *Creating an HTMLFontElement and Setting Its face Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fontObj = bodyObj.createElement("font");
fontObj.face = "courier";
// -->
</script>
</html>
```

HTMLFontElement.size

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFontObj.size`

Description

The size property of the HTMLFontElement object represents the font size.

Example

Listing 11.101 illustrates the creation of an HTMLFontElement using the HTMLBodyElement object element and then setting its size property.

Listing 11.101 *Creating an HTMLFontElement and Setting Its size Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var fontObj = bodyObj.createElement("font");
fontObj.size = 10;
// -->
</script>
</html>
```

HTMLFormElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The FORM element encompasses behavior similar to a collection and an element. It provides direct access to the contained input elements as well as the attributes of the form element. This object inherits all methods and properties from HTMLFormElement. Table 11.12 contains a list of properties and methods for this object.

Table 11.12 *Contents of the HTMLFormElement Object*

Type	Item	Description
Property	acceptCharset	List of character sets supported by the server
	action	Server side form handler
	elements	A collection of all elements present in this form
	enctype	Content type of the submitted form
	length	Number of form controls present in this form

Type	Item	Description
Method	method	HTTP method used to submit form
	name	The name of this form
	target	Frame to render resource in
	reset()	Resets all form controls to their default values
	submit()	Submits this form to the server

Example

Listing 11.102 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then the iteration through its `elements` property.

Listing 11.102 *Creating an `HTMLFormElement` and Iterating Through Its `elements` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var formElements = formObj.elements;
var i = 0;
while(i < formElements.length) {
    processFormElement(formElements.Index(i));
    i++;
}
// -->
</script>
</html>
```

HTMLFormElement.acceptCharset

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.acceptCharset`

Description

The `acceptCharset` property of the `HTMLFormElement` object is a list of character sets supported by the server.

Example

Listing 11.103 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then setting its `acceptCharset` property.

Listing 11.103 *Creating an `HTMLFormElement` and Setting Its `acceptCharset` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.acceptCharset = "us/english";
// -->
</script>
</html>
```

HTMLFormElement.action

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.action`

Description

The action property of the `HTMLFormElement` object is a server-side form handler.

Example

Listing 11.104 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then setting its action property.

Listing 11.104 *Creating an `HTMLFormElement` and Setting Its action Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.action = "http://foo.bar/cgi-bin/processForm.pl";
// -->
</script>
</html>
```

HTMLFormElement.elements

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.elements`

Description

The `elements` property of the `HTMLFormElement` object returns a collection of all control elements in the form.

Example

Listing 11.105 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then the iteration through its `elements` property.

Listing 11.105 *Creating an HTMLFormElement and Iterating Through Its elements Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var formElements = formObj.elements;
var i = 0;
while(i < formElements.length) {
    processFormElement(formElements.item(i));
    i++;
}
// -->
</script>
</html>
```

HTMLFormElement enctype

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.enctype`

Description

The `enctype` property of the `HTMLFormElement` object is the content type of the submitted form, generally "application/x-www-form-urlencoded".

Example

Listing 11.106 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then setting its `enctype` property.

Listing 11.106 *Creating an `HTMLFormElement` and Setting Its `enctype` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.enctype = "application/x-www-form-urlencoded";
// -->
</script>
</html>
```

HTMLFormElement.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.length`

Description

The `length` property of the `HTMLFormElement` object is the number of form controls in the form.

Example

Listing 11.107 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then reading its `length` property.

Listing 11.107 *Creating an `HTMLFormElement` and Reading Its `length` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var formElements = formObj.elements;
var i = 0;
```

```
while(i < formElements.length) {  
    processFormElement(formElements.item(i));  
    i++;  
}  
// -->  
</script>  
</html>
```

HTMLFormElement.method

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFormObj.method

Description

The method property of the HTMLFormElement object is the HTTP method used to submit form.

Example

Listing 11.108 illustrates the creation of an HTMLFormElement using the HTMLBodyElement object and then setting its method property.

Listing 11.108 *Creating an HTMLFormElement and Setting Its method Property*

```
<html>  
<script language="JavaScript" type="text/javascript">  
! --  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var bodyObj = htmlDoc.createElement("body");  
var formObj = bodyObj.createElement("form");  
formObj.method = "post";  
// -->  
</script>  
</html>
```

HTMLFormElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFormObj.name

Description

The name property of the `HTMLFormElement` object names the form.

Example

Listing 11.109 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then setting its name property.

Listing 11.109 *Creating an `HTMLFormElement` and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.name = "checkoutForm";
// -->
</script>
</html>
```

HTMLFormElement.reset()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlFormObj.reset()
```

Description

The `reset()` method of the `HTMLFormElement` object restores a form element's default values. It performs the same action as a reset button.

Example

Listing 11.110 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then the invocation of its `reset()` method.

Listing 11.110 *Creating an `HTMLFormElement` and Invoking Its `reset()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.reset();
```

```
// -->
</script>
</html>
```

HTMLFormElement.submit()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.submit()`

Description

The `submit()` method of the `HTMLFormElement` object submits the form. It performs the same action as a submit button.

Example

Listing 11.111 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then the invocation of its `submit()` method.

Listing 11.111 *Creating an `HTMLFormElement` and Invoking Its `submit()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.submit();
// -->
</script>
</html>
```

HTMLFormElement.target

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFormObj.target`

Description

The `target` property of the `HTMLFormElement` object is the frame to render the resource in.

Example

Listing 11.112 illustrates the creation of an `HTMLFormElement` using the `HTMLBodyElement` object and then setting its `target` property.

Listing 11.112 *Creating an `HTMLFormElement` and Setting Its `target` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
formObj.target = "MainFrame";
// -->
</script>
</html>
```

HTMLFrameElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML frame element. It is used to create a frame. The `HTMLFrameElement` inherits all properties and methods from `HTMLElement`. Table 11.13 contains a list of properties for this object.

Table 11.13 *Properties of the `HTMLFrameElement` Object*

Item	Description
<code>contentDocument</code>	The document this frame contains
<code>frameBorder</code>	The width of the frame borders for the frame
<code>longDesc</code>	URI designating a long description for the frame
<code>marginHeight</code>	Frame margin height in pixels
<code>marginWidth</code>	Frame margin width in pixels
<code>name</code>	The name of this frame
<code>noResize</code>	Indicates whether the user agent can resize this frame
<code>src</code>	URI designating the initial frame contents
<code>scrolling</code>	Indicates whether this frame can be scrolled

Example

Listing 11.113 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object.

Listing 11.113 *Creating an* HTMLFrameElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
// -->
</script>
</html>
```

HTMLFrameElement.contentDocument

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFrameObj.contentDocument

Description

The contentDocument property of the HTMLFrameElement object is the document this frame contains, if there is one and it is available, or Null otherwise.

Example

Listing 11.114 illustrates reading the contentDocument property of a HTMLFrameElement object.

Listing 11.114 *Reading the* contentDocument *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDocObj = frameObj.contentDocument;
// -->
</script>
</html>
```

HTMLFrameElement.frameBorder

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFrameObj.frameBorder

Description

The frameBorder property of the HTMLFrameElement object is the request frame borders.

Example

Listing 11.115 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `frameBorder` property.

Listing 11.115 *Creating an `HTMLFrameElement` and Setting Its `frameBorder` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.Border = 0;
// -->
</script>
</html>
```

HTMLFrameElement.longDesc

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameObj.longDesc`

Description

The `longDesc` property of the `HTMLFrameElement` object is the URI designating a long description of this image or frame.

Example

Listing 11.116 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `longDesc` property.

Listing 11.116 *Creating an `HTMLFrameElement` and Setting Its `longDesc` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.longDesc = "http://foo.bar/framedesc.html";
// -->
</script>
</html>
```

HTMLFrameElement.marginHeight

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameObj.marginHeight`

Description

The `marginHeight` property of the `HTMLFrameElement` object is the frame margin height, in pixels.

Example

Listing 11.117 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `marginHeight` property.

Listing 11.117 *Creating an HTMLFrameElement and Setting Its marginHeight Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.marginHeight = 3;
// -->
</script>
</html>
```

HTMLFrameElement.marginWidth

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameObj.marginWidth`

Description

The `marginWidth` property of the `HTMLFrameElement` object is the frame margin width, in pixels.

Example

Listing 11.118 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `marginWidth` property.

Listing 11.118 *Creating an HTMLFrameElement and Setting Its marginWidth Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.marginWidth = 3;
// -->
</script>
</html>
```

HTMLFrameElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFrameObj.name

Description

The name property of the HTMLFrameElement object is the frame name (object of the target attribute).

Example

Listing 11.119 illustrates the creation of an HTMLFrameElement using the HTMLFrameSetElement object element and then setting its name property.

Listing 11.119 *Creating an HTMLFrameElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.name = "mainframe";
// -->
</script>
</html>
```

HTMLFrameElement.noResize

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFrameObj.noResize

Description

The `noResize` property of the `HTMLFrameElement` object forbids user from resizing frame when true.

Example

Listing 11.120 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `noResize` property.

Listing 11.120 *Creating an `HTMLFrameElement` and Setting Its `noResize` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.noResize = true;
// -->
</script>
</html>
```

HTMLFrameElement.scrolling

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameObj.scrolling`

Description

The `scrolling` property of the `HTMLFrameElement` object specifies whether the frame should have scrollbars.

Example

Listing 11.121 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `scrolling` property.

Listing 11.121 *Creating an `HTMLFrameElement` and Setting Its `scrolling` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.scrolling = false;
// -->
</script>
</html>
```

HTMLFrameElement.src

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameObj.src`

Description

The `src` property of the `HTMLFrameElement` object is a URI designating the initial frame contents.

Example

Listing 11.122 illustrates the creation of an `HTMLFrameElement` using the `HTMLFrameSetElement` object and then setting its `src` property.

Listing 11.122 *Creating an `HTMLFrameElement` and Setting Its `src` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
var frameObj = frameset.createElement("frame");
frameObj.src = "http://foo.bar/ ";
// -->
</script>
</html>
```

HTMLFrameSetElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML frameset element and is used to create a grid of frames. The `HTMLFrameSetElement` object inherits all properties and methods from `HTMLElement`. Table 11.14 contains a list of properties for this object.

Table 11.14 *Properties of the `HTMLFrameSetElement` Object*

Item	Description
<code>cols</code>	The number of columns of frames in a frame set
<code>rows</code>	The number of rows of frames in a frame set

Example

Listing 11.123 illustrates the creation of an `HTMLFrameSetElement` using the `HTMLDocument` object.

Listing 11.123 *Creating an* `HTMLFrameSetElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");

// -->
</script>
</html>
```

HTMLFrameSetElement.cols

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlFrameSetObj.cols`

Description

The `cols` property of the `HTMLFrameSetElement` object is the number of columns of frames in the frameset.

Example

Listing 11.124 illustrates the creation of an `HTMLFrameSetElement` using the `HTMLDocument` object and then setting its `cols` property.

Listing 11.124 *Creating an* `HTMLFrameSetElement` *and Setting Its* `cols` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
frameset.cols = 3;

// -->
</script>
</html>
```

HTMLFrameSetElement.rows

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlFrameSetObj.rows

Description

The rows property of the HTMLFrameSetElement object is the number of rows of frames in the frameset.

Example

Listing 11.125 illustrates the creation of an HTMLFrameSetElement using the HTMLDocument object element and then setting its rows property.

Listing 11.125 *Creating an HTMLFrameSetElement and Setting Its rows Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
frameSetObj = htmlDoc.createElement("frameset");
frameset.rows = 3;
// -->
</script>
</html>
```

HTMLHeadElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents this document's header information. This object inherits all methods and properties from the HTMLElement object. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
profile	URI specifying the metadata profile

Example

Listing 11.126 illustrates the creation of an HTMLHeadElement using the HTMLDocument object.

Listing 11.126 *Creating an* HTMLHeadElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var headElement = htmlDoc.createElement("head");

// -->
</script>
</html>
```

HTMLHeadElement.profile

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlHeadElemObj.profile

Description

The profile property of the HTMLHeadElement object represents the URI designating a metadata profile.

Example

Listing 11.127 illustrates the creation of an HTMLHeadElement using the HTMLDocument object element and then setting its profile property.

Listing 11.127 *Creating an* HTMLHeadElement *and Setting Its* profile Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var headElement = htmlDoc.createElement("head");
headElement.profile = "http://foo.bar/metaprofile.htm";
// -->
</script>
</html>
```

HTMLHeadingElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents HTML heading elements H1 to H6. The HTMLHeadingElement object inherits all properties and methods from the HTMLElement object. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
align	The horizontal alignment for this heading element

Example

Listing 11.128 illustrates the creation of an HTMLHeadingElement using the HTMLBodyElement object element and then setting its align property.

Listing 11.128 *Creating an HTMLHeadingElement and Setting Its align Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var headingObj = bodyObj.createElement("heading");
headingObj.align = "left";
// -->
</script>
</html>
```

HTMLHeadingElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlHeadingingObj.align

Description

The align property of the HTMLHeadingElement object is the horizontal text alignment.

Example

Listing 11.129 illustrates the creation of an HTMLHeadingElement using the HTMLBodyElement object and then setting its align property.

Listing 11.129 *Creating an `HTMLHeadingElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var headingObj = bodyObj.createElement("heading");
headingObj.align = "bottom";
// -->
</script>
</html>
```

HTMLHRElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML horizontal rule element. The `HTMLHRElement` inherits all properties and methods from `HTMLElement`. Table 11.15 contains a list of properties for this object.

Table 11.15 *Properties of the `HTMLHRElement` Object*

Item	Description
<code>align</code>	Indicates the alignment of the rule element on the page
<code>noShade</code>	Indicates to the client that there should be no shade in this element's rendering
<code>size</code>	The height of this horizontal rule element
<code>width</code>	The width of this horizontal rule element

Example

Listing 11.130 illustrates the creation of an `HTMLHRElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.130 *Creating an `HTMLHRElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var hrObj = bodyObj.CreateElement("hr");
hrObj.align = "top";
// -->
</script>
</html>
```

HTMLHRElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlHRObj.align`

Description

The `align` property of the `HTMLHRElement` object aligns the rule on the page.

Example

Listing 11.131 illustrates the creation of an `HTMLHRElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.131 Creating an HTMLHRElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var hrObj = bodyObj.CreateElement("hr");
hrObj.align = "bottom";
// -->
</html>
```

HTMLHRElement.noShade

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlHRObj.noShade`

Description

The `noShade` property of the `HTMLHRElement` object indicates to the user agent that there should be no shading in the rendering of this element.

Example

Listing 11.132 illustrates the creation of an `HTMLHRElement` using the `HTMLBodyElement` object and then setting its `noShade` property.

Listing 11.132 Creating an HTMLHRElement and Setting Its noShade Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var hrObj = bodyObj.CreateElement("hr");
hrObj.noShade = true;
```

```
// -->
</script>
</html>
```

HTMLHRElement.size

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlHRObj.size`

Description

The size property of the HTMLHRElement object is the height of the rule.

Example

Listing 11.133 illustrates the creation of an HTMLHRElement using the HTMLBodyElement object and then setting its size property.

Listing 11.133 *Creating an HTMLHRElement and Setting Its size Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var hrObj = bodyObj.CreateElement("hr");
hrObj.size = 10;
// -->
</script>
</html>
```

HTMLHRElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlHRObj.width`

Description

The width property of the HTMLHRElement object is the width of the horizontal rule.

Example

Listing 11.134 illustrates the creation of an HTMLHRElement using the HTMLBodyElement object element and then setting its width property.

Listing 11.134 *Creating an HTMLHRElement and Setting Its width Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var hrObj = bodyObj.CreateElement("hr");
hrObj.width = 10;
// -->
</script>
</html>
```

HTMLHtmlElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents the root of an HTML document. HTMLHtmlElement inherits all properties and methods of the HTMLDocument object. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
version	Version information about this document's DTD

Example

Listing 11.135 illustrates the creation of an HTMLHtmlElement using the HTMLDocument object element.

Listing 11.135 *Creating an HTMLHtmlElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHtmlDocument("My HTML Doc");
htmlDoc.open();
var htmlElement = htmlDoc.createElement("html");

// -->
</script>
</html>
```

HTMLHtmlElement.version

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlHtmlElementObj.version`

Description

The `version` property of the `HTMLHtmlElement` represents version information about the document's DTD.

Example

Listing 11.136 illustrates the creation of an `HTMLHtmlElement` using the `HTMLDocument` object element and then setting its `version` property.

Listing 11.136 *Creating an HTMLHtmlElement and Setting Its version Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHtmlDocument("My HTML Doc");
htmlDoc.open();
var htmlElement = htmlDoc.createElement("html");
htmlElement.version = "4.0";

// -->
</script>
</html>
```

HTMLIFrameElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML internal frame element. The `HTMLIFrameElement` object inherits all properties and methods of the `HTMLElement` object. Table 11.16 contains a list of properties for this object.

Table 11.16 Properties of the HTMLIFrameElement Object

Item	Description
align	Aligns this IFRAME in regard to surrounding text
contentDocument	The document this frame contains
frameBorder	Indicates the width of the frame border for this IFRAME
height	Indicates the height for this IFRAME
longDesc	URI designating the location of a long description for this IFRAME
marginHeight	Frame margin height in pixels
marginWidth	Frame margin width in pixels
name	The name of this IFRAME
scrolling	Indicates whether this frame can be scrolled
src	URI designating initial frame contents
width	Width of this IFRAME

Example

Listing 11.137 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement.

Listing 11.137 Creating an HTMLIFrameElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");

// -->
</script>
</html>
```

HTMLIFrameElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlIFrameObj.align
```

Description

The align property of the HTMLFrameElement object aligns this object (vertically or horizontally) with respect to its surrounding text.

Example

Listing 11.138 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its align property.

Listing 11.138 *Creating an HTMLIFrameElement and Setting Its align Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.align = "top";
// -->
</script>
</html>
```

HTMLIFrameElement.contentDocument

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlIFrameObj.contentDocument`

Description

The `contentDocument` property of the `HTMLIFrameElement` object represents the document this frame contains, if there is one and it is available, or `Null` otherwise.

Example

Listing 11.139 illustrates reading an `HTMLIFrameElement`'s `contentDocument` property.

Listing 11.139 *Reading an HTMLIFrameElement's contentDocument Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = iFrameObj.contentDocument;
// -->
</script>
</html>
```

HTMLIFrameElement.frameBorder

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlIFrameObj.frameBorder`

Description

The `frameBorder` property of the `HTMLIFrameElement` object requests the frame borders for this object.

Example

Listing 11.140 illustrates the creation of an `HTMLIFrameElement` using the `HTMLBodyElement` object and then setting its `frameBorder` property.

Listing 11.140 *Creating an `HTMLIFrameElement` and Setting Its `frameBorder` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.frameBorder = 1;
// -->
</script>
</html>
```

HTMLIFrameElement.height

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlIFrameObj.height`

Description

The `height` property of the `HTMLIFrameElement` object represents the frame height.

Example

Listing 11.141 illustrates the creation of an `HTMLIFrameElement` using the `HTMLBodyElement` object and then setting its `height` property.

Listing 11.141 *Creating an `HTMLIFrameElement` and Setting Its `height` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.height = 25;
// -->
</script>
</html>
```

HTMLIFrameElement.longDesc

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlIFrameObj.longDesc`

Description

The `longDesc` property of the `HTMLIFrameElement` object represents the URI designating a long description of this image or frame.

Example

Listing 11.142 illustrates the creation of an `HTMLIFrameElement` using the `HTMLBodyElement` object and then setting its `longDesc` property.

Listing 11.142 *Creating an `HTMLIFrameElement` and Setting Its `longDesc` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.longDesc = "http://foo.bar/desc.html";
// -->
</script>
</html>
```

HTMLIFrameElement.marginHeight

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlIFrameObj.marginHeight`

Description

The `marginHeight` property of the `HTMLIFrameObject` represents the frame margin height, in pixels.

Example

Listing 11.143 illustrates the creation of an `HTMLIFrameElement` using the `HTMLBodyElement` object and then setting its `marginHeight` property.

Listing 11.143 *Creating an HTMLIFrameElement and Setting the marginHeight Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.marginHeight = 4;
// -->
</script>
</html>
```

HTMLIFrameElement.marginWidth

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIFrameObj.marginWidth

Description

The marginWidth property of the HTMLIFrameElement represents the frame margin width, in pixels.

Example

Listing 11.144 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its marginWidth property.

Listing 11.144 *Creating an HTMLIFrameElement and Setting the marginWidth Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.width = 33;
// -->
</script>
</html>
```

HTMLIFrameElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIFrameObj.name

Description

The name property of the HTMLIFrameElement object represents the frame name (object of the target attribute).

Example

Listing 11.145 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its name property.

Listing 11.145 *Creating an HTMLIFrameElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.name = "myframe";
// -->
</script>
</html>
```

HTMLIFrameElement.scrolling

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIFrameObj.scrolling

Description

The scrolling property of the HTMLIFrameElement object specifies whether the frame should have scrollbars.

Example

Listing 11.146 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its scrolling property.

Listing 11.146 *Creating an HTMLIFrameElement and Setting Its scrolling Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.scrolling = false;
// -->
</script>
</html>
```

HTMLIFrameElement.src

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIFrameObj.src

Description

The src property of the HTMLIFrameElement object represent a URI designating the initial frame contents.

Example

Listing 11.147 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its src property.

Listing 11.147 Creating an HTMLIFrameElement and Setting Its src Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var iFrameObj = bodyObj.createElement("iframe");
iFrameObj.src = "http://foo.bar/";
// -->
</script>
</html>
```

HTMLIFrameElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIFrameObj.width

Description

The width property of the HTMLIFrameElement object represents the frame width.

Example

Listing 11.148 illustrates the creation of an HTMLIFrameElement using the HTMLBodyElement object and then setting its width property.

Listing 11.148 Creating an HTMLIFrameElement and Setting Its width Property

```
<html>
<script language="JavaScript" type="text/javascript">
```

```
<!--  
var iFrameObj = bodyObj.createElement("iframe");  
iFrameObj.width = 100;  
// -->  
</script>  
</html>
```

HTMLImageElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML image element. The `HTMLImageElement` object inherits all methods and properties from the `HTMLElement` object. Table 11.17 contains a list of properties for this object.

Table 11.17 *Properties of the* `HTMLImageElement` *Object*

Item	Description
<code>align</code>	Aligns image with surrounding text
<code>alt</code>	Alternate text for image
<code>border</code>	Width of border around image
<code>height</code>	The overridden height of the image element
<code>hspace</code>	Horizontal space to the right and left of this image element
<code>isMap</code>	Indicates whether this image uses a client side image map
<code>longDesc</code>	URI designating a long description for this image
<code>lowSrc</code>	URI indicating a low resolution alternate image
<code>name</code>	The name of this image element
<code>src</code>	URI designating the image file for this image element
<code>useMap</code>	Indicates the name of an image map to use for this image element
<code>vspace</code>	The space between the image and the top and bottom of this image element
<code>width</code>	The overridden width of this image element

Example

Listing 11.149 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `lowSrc` property.

Listing 11.149 *Creating an `HTMLImageElement` and Setting Its `lowSrc` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.lowSrc = "http://foo.bar/images/lowfi.jpg";
// -->
</script>
</html>
```

HTMLImageElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.align`

Description

The `align` property of the `HTMLImageElement` object aligns this object (vertically or horizontally) with respect to its surrounding text.

Example

Listing 11.150 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.150 *Creating an `HTMLImageElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("image");
imageObj.align = "top";
// -->
</script>
</html>
```

HTMLImageElement.alt

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.alt`

Description

The `alt` property of the `HTMLImageElement` object represents alternate text for user agents not rendering the normal content of this element.

Example

Listing 11.151 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `alt` property.

Listing 11.151 *Creating an `HTMLImageElement` and Setting Its `alt` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.CreateElement("image");
imageObj.alt = "Can't support images?";
// -->
</script>
</html>
```

HTMLImageElement.border

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.border`

Description

The `border` property of the `HTMLImageElement` indicates the width of border around image.

Example

Listing 11.152 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `border` property.

Listing 11.152 *Creating an `HTMLImageElement` and Setting Its `border` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("image");
imageObj.border = 1;
// -->
</script>
</html>
```

HTMLImageElement.height

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.height`

Description

The height property of the HTMLImageElement object overrides the height of the image.

Example

Listing 11.153 illustrates the creation of an HTMLImageElement using the HTMLBodyElement object element and then setting its height property.

Listing 11.153 *Creating an HTMLImageElement and Setting Its height Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.height = 12;
// -->
</script>
</html>
```

HTMLImageElement.hspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.hspace`

Description

The hspace property of the HTMLImageElement object indicates the horizontal space to the left and right of this image.

Example

Listing 11.154 illustrates the creation of an HTMLImageElement using the HTMLBodyElement object and then setting its hspace property.

Listing 11.154 *Creating an `HTMLImageElement` and Setting Its `hspace` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.hspace = 12;
// -->
</script>
```

HTMLImageElement.isMap

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.isMap`

Description

The `isMap` property of the `HTMLImageElement` object indicates whether or not to use a server-side image map.

Example

Listing 11.155 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `isMap` property.

Listing 11.155 *Creating an `HTMLImageElement` and Setting Its `isMap` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.isMap = false;
// -->
</script>
</html>
```

HTMLImageElement.longDesc

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.longDesc`

Description

The `longDesc` property of the `HTMLImageElement` object indicates the URI designating a long description of this image or frame.

Example

Listing 11.156 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `longDesc` property.

Listing 11.156 Creating an `HTMLImageElement` and Setting Its `longDesc` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.longDesc = "This is a long description of this image.";
// -->
</script>
</html>
```

HTMLImageElement.lowSrc

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlImageObj.lowSrc`

Description

The `lowSrc` property of the `HTMLImageElement` object indicates the URI designating the source of this image for low-resolution output.

Example

Listing 11.157 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object element and then setting its `lowSrc` property.

Listing 11.157 Creating an `HTMLImageElement` and Setting Its `lowSrc` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.lowSrc = "http://foo.bar/images/lowfi.jpg";
// -->
</script>
</html>
```

HTMLImageElement.src

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlImageObj.src

Description

The src property of the HTMLImageElement object indicates the URI designating the source of this image.

Example

Listing 11.158 illustrates the creation of an HTMLImageElement using the HTMLBodyElement object element and then setting its src property.

Listing 11.158 *Creating an HTMLImageElement and Setting Its src Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.src = "http://foo.bar/images/foo.jpg";
// -->
</script>
</html>
```

HTMLImageElement.useMap

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlImageObj.useMap

Description

The useMap property of the HTMLImageElement object indicates the name of the client-side image map to use.

Example

Listing 11.159 illustrates the creation of an HTMLImageElement using the HTMLBodyElement object element and then setting its useMap property.

Listing 11.159 *Creating an HTMLImageElement and Setting Its useMap Property*

```
<html>
<script language="JavaScript" type="text/javascript">
```

Listing 11.159 Continued

```
<!--  
var imageObj = bodyObj.createElement("img");  
imageObj.useMap = "myClientMap";  
// -->  
</script>  
</html>
```

HTMLImageElement.vspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlImageObj.vspace

Description

The *vspace* property of the *HTMLImageElement* object indicates the vertical space above and below this image.

Example

Listing 11.160 illustrates the creation of an *HTMLImageElement* using the *HTMLBodyElement* object and then setting its *vspace* property.

Listing 11.160 Creating an HTMLImageElement and Setting Its vspace Property

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var imageObj = bodyObj.createElement("img");  
imageObj.vspace = 2;  
// -->  
</script>  
</html>
```

HTMLImageElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlImageObj.width

Description

The *width* property of the *HTMLImageElement* object overrides width of the image.

Example

Listing 11.161 illustrates the creation of an `HTMLImageElement` using the `HTMLBodyElement` object and then setting its `width` property.

Listing 11.161 *Creating an `HTMLImageElement` and Setting Its `width` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var imageObj = bodyObj.createElement("img");
imageObj.width = 23;
// -->
</script>
</html>
```

HTMLInputElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents a form control used for text input.

NOTE

Depending on the environment in which the page is being viewed, the `value` property might be read-only for the file upload input type. For the password input type, the actual value returned might be masked to prevent unauthorized use.

This object inherits all methods and properties of the `HTMLElement` object.

Table 11.18 contains a list of properties and methods for this object.

Table 11.18 *Contents of the `HTMLInputElement` Object*

Type	Item	Description
Property	<code>accept</code>	List of content type that the server will handle correctly.
	<code>accessKey</code>	Single character access key that gives control to this input element.
	<code>align</code>	Aligns object with respect to surrounding text.
	<code>alt</code>	Alternate text for clients not able to display input elements.
	<code>checked</code>	Indicates whether this input element is checked.

Table 11.18 Continued

Type	Item	Description
Method	defaultChecked	Indicates if this input element is checked by default.
	defaultValue	The default text value for this input element.
	disabled	Indicates whether this input element is disabled.
	form	The form in which this input element resides.
	maxLength	Indicates the maximum character length for this input element.
	name	The name of this input element.
	readOnly	Indicates whether this input element's contents are readonly.
	size	Indicates the number of characters that will be visible in this input element.
	src	When type equals "image", this will be the URI to the image file.
	tabIndex	Number representing the tab order in this document.
	type	The type of this input element; that is, radio, text.
	useMap	Specifies a client-side image map to use.
	value	The value of this input element.
	blur()	Removes keyboard focus from this input element.
	click()	Simulates a mouse click on this input element.
	focus()	Gives keyboard focus to this input element.
	select()	Selects the contents of this input element.

Example

Listing 11.162 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement`.

Listing 11.162 Creating an `HTMLInputElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
// -->
</script>
</html>
```


HTMLInputElement.accept

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.accept

Description

The accept property of the HTMLInputElement object is a comma-separated list of content types that a server processing this form will handle correctly.

Example

Listing 11.163 illustrates the accept property of an HTMLInputElement object.

Listing 11.163 *Setting the accept Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
inputObj.accept = "text/html";
// -->
</script>
</html>
```

HTMLInputElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.accessKey

Description

The accessKey property of the HTMLInputElement object is a single character access key to give access to the form control.

Example

Listing 11.164 illustrates the creation of an HTMLInputElement using the HTMLFormElement object element and then setting its accessKey property.

Listing 11.164 *Creating an HTMLInputElement and Setting Its accessKey Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

Listing 11.164 Continued

```
var inputObj = formObj.createElement("input");
inputObj.accessKey = "A";
// -->
</html>
```

HTMLInputElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.align

Description

The align property of the HTMLInputElement object aligns this object (vertically or horizontally) with respect to its surrounding text.

Example

Listing 11.165 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then setting its align property.

Listing 11.165 Creating an HTMLInputElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.align = "left";
// -->
</script>
</html>
```

HTMLInputElement.alt

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.alt

Description

The alt property of the HTMLInputElement object is alternate text for user agents not rendering the normal content of this element.

Example

Listing 11.166 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object element and then setting its `alt` property.

Listing 11.166 *Creating an `HTMLInputElement` and Setting Its `alt` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.alt = "Input here!";
// -->
</html>
```

HTMLInputElement.blur()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.blur()`

Description

The `blur()` method of the `HTMLInputElement` object removes keyboard focus from this element.

Example

Listing 11.167 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then the invocation of its `blur()` method.

Listing 11.167 *Creating an `HTMLInputElement` and Invoking Its `blur()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.blur();
// -->
</script>
</html>
```

HTMLInputElement.checked

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.checked`

Description

The checked property of the `HTMLInputElement` object is relevant when the type attribute of the element has the value "Radio" or "Checkbox", this represents the current state of the form control, in an interactive user agent. Changes to this attribute change the state of the form control, but do not change the value of the HTML value attribute of the element.

Example

Listing 11.168 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its checked property.

Listing 11.168 *Creating an `HTMLInputElement` and Setting Its checked Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.Type = "checkbox";
input.checked = false;
// -->
</script>
</html>
```

HTMLInputElement.click()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlInputObj.click()
```

Description

The `click()` method of the `HTMLInputElement` object simulates a mouse click. For INPUT elements whose type attribute has one of the following values: "Button", "Checkbox", "Radio", "Reset", or "Submit".

Example

Listing 11.169 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then the invocation of its `click()` method.

Listing 11.169 *Creating an `HTMLInputElement` and Invoking Its click() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

```
var inputObj = formObj.createElement("input");
inputObj.type = "checkbox";
inputObj.click();
// -->
</html>
```

HTMLInputElement.defaultChecked

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.defaultChecked

Description

The `defaultChecked` property of the `HTMLInputElement` object is relevant when `type` has the value "Radio" or "Checkbox", this represents the HTML checked attribute of the element. The value of this attribute doesn't change if the state of the corresponding form control, in an interactive user agent, changes. Changes to this attribute, however, resets the state of the form control.

Example

Listing 11.170 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its `defaultChecked` property.

Listing 11.170 *Creating an HTMLInputElement and Setting Its defaultChecked Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "checkbox";
inputObj.defaultChecked = true;
// -->
</html>
```

HTMLInputElement.defaultValue

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.defaultValue

Description

The `defaultValue` property of the `HTMLInputElement` object is relevant when the `type` attribute of the element has the value "Text", "File", or "Password", this represents the HTML value attribute of the element. The value of this attribute doesn't change if the contents of the corresponding form control, in an interactive user agent, changes. Changing this attribute, however, resets the contents of the form control.

Example

Listing 11.171 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its `defaultValue` property.

Listing 11.171 *Creating an `HTMLInputElement` and Setting Its `defaultValue` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
input.type = "Password";
input.defaultValue = "";
// -->
</script>
</html>
```

HTMLInputElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.disabled`

Description

The `disabled` property of the `HTMLInputElement` object is used to disable the input object instance.

Example

Listing 11.172 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its `disabled` property.

Listing 11.172 *Creating an `HTMLInputElement` and Setting Its `disabled` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
```

```
inputObj.disabled = true;  
// -->  
</html>
```

HTMLInputElement.focus()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.focus()

Description

The `focus()` method of the `HTMLInputElement` object gives keyboard focus to this element.

Example

Listing 11.173 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object element and then the invocation of its `focus()` method.

Listing 11.173 *Creating an `HTMLInputElement` and Invoking Its `focus()` Method*

```
<html>  
<script language="JavaScript" type="text/javascript">  
!- -  
var inputObj = formObj.createElement("input");  
inputObj.focus();  
// -->  
</html>
```

HTMLInputElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.form

Description

The `form` property of the `HTMLInputElement` object returns the `FORM` element containing this control. Returns `Null` if this control isn't within the context of a form.

Example

Listing 11.174 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then reading its `form` property.

Listing 11.174 *Creating an HTMLInputElement and Reading Its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
if(inputObj.form.name == "emailForm")
    processEmailForm(inputObj.form);
// -->
</html>
```

HTMLInputElement.maxLength

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.maxLength

Description

The `maxLength` property of the `HTMLInputElement` object is the maximum number of characters for text fields, when type has the value "Text" or "Password".

Example

Listing 11.175 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its `maxLength` property.

Listing 11.175 *Creating an HTMLInputElement and Setting Its maxLength Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
inputObj.maxLength = 25;
// -->
</html>
```

HTMLInputElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.name

Description

The name property of the HTMLInputElement object is the form control or object name when submitted with a form.

Example

Listing 11.176 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then setting its name property.

Listing 11.176 *Creating an HTMLInputElement and Setting name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.name = "address";
// -->
</html>
```

HTMLInputElement.readOnly

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.readOnly

Description

The readOnly property of the HTMLInputElement object indicates that this control is read-only. Relevant only when type has the value "Text" or "Password".

Example

Listing 11.177 illustrates the creation of an HTMLInputElement using the HTMLFormElement object element and then setting its readOnly property.

Listing 11.177 *Creating an HTMLInputElement and Setting Its readOnly Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
inputObj.value = "readonly field";
inputObj.readOnly = true;
// -->
</html>
```

HTMLInputElement.select()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.select()

Description

The select() method of the HTMLInputElement object selects the contents of the text area. For INPUT elements whose type attribute has one of the following values: "Text", "File", or "Password".

Example

Listing 11.178 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then the invocation of its select() method.

Listing 11.178 *Creating an HTMLInputElement and Invoking Its select() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
inputObj.value = "foo bar";
inputObj.select();
// -->
</html>
```

HTMLInputElement.size

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.size

Description

The size property of the HTMLInputElement object indicates the size information. The precise meaning is specific to each type of field.

Example

Listing 11.179 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then setting its size property.

Listing 11.179 *Creating an `HTMLInputElement` and Setting Its `size` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
inputObj.size = 25;
// -->
</html>
```

HTMLInputElement.src

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.src`

Description

The `src` property of the `HTMLInputElement` object is relevant when the `type` attribute has the value "Image", this attribute specifies the location of the image to be used to decorate the graphical submit button.

Example

Listing 11.180 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object element and then setting its `src` property.

Listing 11.180 *Creating an `HTMLInputElement` and Setting Its `src` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "image";
inputObj.src = "http://foo.bar/images/foo.jpg";
// -->
</html>
```

HTMLInputElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.tabIndex`

Description

The `tabIndex` property of the `HTMLInputElement` object is the index that represents the element's position in the tabbing order.

Example

Listing 11.181 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object and then setting its `tabIndex` property.

Listing 11.181 Creating an `HTMLInputElement` and Setting Its `tabIndex` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
inputObj.tabIndex = 1;
// -->
</html>
```

HTMLInputElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlInputObj.type`

Description

The `type` property of the `HTMLInputElement` object is the type of control created.

Example

Listing 11.182 illustrates the creation of an `HTMLInputElement` using the `HTMLFormElement` object element and then setting its `type` property.

Listing 11.182 Creating an `HTMLInputElement` and Setting Its `type` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "text";
// -->
</html>
```

HTMLInputElement.useMap

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.useMap

Description

The useMap property of the HTMLInputElement object indicates the name of the client-side image map to use.

Example

Listing 11.183 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then setting its useMap property.

Listing 11.183 *Creating an HTMLInputElement and Setting Its useMap Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "image";
inputObj.useMap = "mymap";
// -->
</html>
```

HTMLInputElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlInputObj.value

Description

The value property of the HTMLInputElement object is relevant when the type attribute of the element has the value "Text", "File", or "Password", this represents the current contents of the corresponding form control, in an interactive user agent. Changing this attribute changes the contents of the form control, but doesn't change the value of the HTML value attribute of the element. When the type attribute of the element has the value "Button", "Hidden", "Submit", "Reset", "Image", "Checkbox", or "Radio", this represents the HTML value attribute of the element.

Example

Listing 11.184 illustrates the creation of an HTMLInputElement using the HTMLFormElement object and then setting its value property.

Listing 11.184 *Creating an* `HTMLInputElement` *and Setting Its* value Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var inputObj = formObj.createElement("input");
inputObj.type = "reset";
inputObj.value = "Clear form values";
// -->
</html>
```

HTMLIsIndexElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This element is used for single-line text input. `HTMLIsIndexElement` inherits all methods and properties from `HTMLElement`. Table 11.19 contains a list of properties for this object.

Table 11.19 *Properties of* `HTMLIsIndexElement` *Object*

Item	Description
form	Returns the form element containing this element
prompt	A prompt message for this element

Example

Listing 11.185 illustrates the creation of an `HTMLIsIndexElement` using the `HTMLDocument` object.

Listing 11.185 *Creating an* `HTMLIsIndexElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var isIndexElement = htmlDoc.createElement("isindex");

}
</html>
```

HTMLIsIndexElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIsIndexObj.form

Description

The form property of the HTMLIsIndexElement object returns the FORM element containing this control. Returns Null if this control isn't within the context of a form.

Example

Listing 11.186 illustrates the creation of an HTMLIsIndexElement using the HTMLFormElement object element and then reading its form property.

Listing 11.186 *Creating an HTMLIsIndexElement and Reading Its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHtmlDocument("My HTML Doc");
htmlDoc.open();
var isIndexElement = htmlDoc.createElement("isIndex");
if(isIndexElement.form == "emailForm")
    processIsIndex(isIndexElement);
}
// -->
</script>

var isIndexElement = htmlDoc.createElement("isIndex");
</html>
```

HTMLIsIndexElement.prompt

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlIsIndexObj.prompt

Description

The prompt property of the HTMLIsIndexElement object is the prompt message.

Example

Listing 11.187 illustrates the creation of an HTMLIsIndexElement using the HTMLDocument object element and then setting its prompt properties.

Listing 11.187 *Creating an HTMLIsIndexElement and Setting Its prompt Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var isIndexElement = htmlDoc.createElement("isindex");
isIndexElement.prompt = "prompt";
}
</script>
</html>
```

HTMLLabelElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML form field text label. This object inherits all methods and properties of the HTMLInputElement object. Table 11.20 contains a list of properties for this object.

Table 11.20 *Properties of HTMLLabelElement Object*

Item	Description
form	The form in which this label element resides
accessKey	Single character access key that gives control to this label element
htmlFor	Links this label to another element by id

Example

Listing 11.188 illustrates the creation of an HTMLLabelElement using the HTMLFormElement object.

Listing 11.188 *Creating an HTMLLabelElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var labelObj = formObj.createElement("label");

// -->
</html>
```


HTMLLabelElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLabelObj.accessKey

Description

The accessKey property of the HTMLLabelElement object is a single character access key to give access to the form control.

Example

Listing 11.189 illustrates the creation of an HTMLLabelElement using the HTMLBodyElement object and then setting its accessKey property.

Listing 11.189 *Creating an HTMLLabelElement and Setting Its accessKey Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var labelObj = formObj.createElement("label");
labelObj.accessKey = "L";
// -->
</html>
```

HTMLLabelElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLabelObj.form

Description

The form property of the HTMLLabelElement returns the FORM element containing this control. Returns Null if this control isn't within the context of a form.

Example

Listing 11.190 illustrates the creation of an HTMLLabelElement using the HTMLFormElement object and then reading its form property.

Listing 11.190 *Creating an HTMLLabelElement and Reading Its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var labelObj = formObj.createElement("label");
if(labelObj.form.Name == "emailForm")
    processEmailForm(labelObj.form);
// -->
</html>
```

HTMLLabelElement.htmlFor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLabelObj.htmlFor

Description

The htmlFor property of the HTMLLabelElement object links this label with another form control by the id attribute.

Example

Listing 11.191 illustrates the creation of an HTMLLabelElement using the HTMLFormElement object element and then setting its htmlFor property.

Listing 11.191 *Creating an HTMLLabelElement and Setting Its htmlFor Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var labelObj = formObj.createElement("label");
var inputObj = formObj.createElement("input");
inputObj.name = "address";
labelObj.htmlFor = "address";
// -->
</html>
```

HTMLLegendElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents a caption for a FIELDSET grouping. HTMLLegendElement inherits all properties and methods of the HTMLElement object. Table 11.21 contains a list of properties for this object.

Table 11.21 *Properties of the HTMLLegendElement Object*

Item	Description
accessKey	A single character access key that gives control to this legend element
align	Text alignment relative to the corresponding FIELDSET
form	The form in which this legend element resides

Example

Listing 11.192 illustrates the creation of an HTMLLegendElement using the HTMLFormElement object.

Listing 11.192 *Creating an HTMLLegendElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var legendObj = formObj.createElement("legend");
// -->
</html>
```

HTMLLegendElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLegendObj.accessKey

Description

The accessKey property of the HTMLLegendElement object is a single character access key to give access to the form control.

Example

Listing 11.193 illustrates the creation of an HTMLLegendElement using the HTMLFormElement object and then setting its accessKey property.

Listing 11.193 *Creating an HTMLLegendElement and Setting Its accessKey Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var legendObj = formObj.createElement("legend");
legendObj.accessKey = "L";
// -->
</html>
```

HTMLLegendElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLegendObj.align

Description

The align property of the HTMLLegendElement object is the text alignment relative to FIELDSET.

Example

Listing 11.194 illustrates the creation of an HTMLLegendElement using the HTMLFormElement object and then setting its align property.

Listing 11.194 *Creating an HTMLLegendElement and Setting align Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var legendObj = formObj.createElement("legend");
legendObj.align = "top";
// -->
</html>
```

HTMLLegendElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLegendObj.form

Description

The form property of the HTMLLegendElement object returns the FORM element containing this control. Returns Null if this control isn't within the context of a form.

Example

Listing 11.195 illustrates the creation of an `HTMLLegendElement` using the `HTMLFormElement` object and then reading its `form` property.

Listing 11.195 *Creating an `HTMLLegendElement` and Reading Its `form` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var legendObj = formObj.createElement("legend");
if(legendObj.form.Name == "emailForm")
    processEmailForm(legendObj.form);
// -->
</html>
```

HTMLLIElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML list item element. `HTMLLIElement` inherits all properties and methods of `HTMLElement`. Table 11.22 contains a list of properties for this object.

Table 11.22 *Properties of the `HTMLLIElement` Object*

Item	Description
<code>type</code>	List item element bullet style
<code>value</code>	Contents of this list item element

Example

Listing 11.196 illustrates the creation of an `HTMLLIElement` using the `HTMLLOElement` object.

Listing 11.196 *Creating an `HTMLLIElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElment("ol");
var liObj = olObj.createElement("li");
```

Listing 11.196 Continued

```
liObj.type = "item";  
// -->  
</script>  
</html>
```

HTMLLIElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLIObj.type

Description

The type property of the HTMLLIElement object is the list item bullet style.

Example

Listing 11.197 illustrates the creation of an HTMLLIElement using the HTMLOLElement object and then setting its type property.

Listing 11.197 Creating an HTMLLIElement and Setting Its type Property

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var o1Obj = bodyObj.CreateElement("ol");  
var liObj = o1Obj.createElement("li");  
  
liObj.type = "item";  
// -->  
</script>  
</html>
```

HTMLLIElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLIObj.value

Description

The value property of the HTMLLIElement object resets the sequence number when used in OL.

Example

Listing 11.198 illustrates the creation of an `HTMLLIElement` using the `HTMLOLElement` object and then setting its `value` property.

Listing 11.198 *Creating an `HTMLLIElement` and Setting Its `value` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElement("ol");
var liObj = olObj.createElement("li");

liObj.value = "list item text";
// -->
</script>
</html>
```

HTMMLinkElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The `LINK` element specifies a link to an external resource, and defines this document's relationship to that resource (or vice versa). Table 11.23 contains a list of properties for this object.

Table 11.23 *Properties of the `HTMMLinkElement` Object*

Item	Description
<code>charset</code>	Indicates the character encoding of this element
<code>disabled</code>	Indicates whether this link is disabled
<code>href</code>	URI of the linked resource
<code>hreflang</code>	Language code of the linked resource
<code>media</code>	Media type of the linked resource
<code>rel</code>	Forward link type
<code>rev</code>	Reverse link type
<code>target</code>	Frame to render resource in
<code>type</code>	Advisory content type

Example

Listing 11.199 illustrates the creation of an `HTMMLinkElement` using the `HTMLDocument` object element and then setting its `href` property.

Listing 11.199 *Creating an* `HTMLLinkElement` *and Setting Its* `href` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.href = "http://foo.bar/ ";
}
</html>
```

HTMLLinkElement.charset

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlLinkObj.charset`

Description

The `charset` property of the `HTMLLinkElement` object is the character encoding of the resource being linked to.

Example

Listing 11.200 illustrates the creation of and displaying the `charset` property of a `HTMLLinkElement` object.

Listing 11.200 *Displaying the* `charset` *Property of* `HTMLLinkElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
htmlDoc.write(linkElement.charset);
}
</html>
```

HTMLLinkElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlLinkObj.disabled`

Description

The disabled property of the HTMLLinkElement enables/disables the link. This is currently only used for style sheet links, and can be used to activate or deactivate style sheets.

Example

Listing 11.201 illustrates the creation of an HTMLLinkElement using the HTMLDocument object element and then setting its disabled property to true.

Listing 11.201 *Creating an HTMLLinkElement and Setting Its disabled Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.disabled = true;
}
</html>
```

HTMLLinkElement.href

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLinkObj.href

Description

The href property of the HTMLLinkElement object is the URI of the linked resource.

Example

Listing 11.202 illustrates the creation of an HTMLLinkElement using the HTMLDocument object element and then setting its href property.

Listing 11.202 *Creating an HTMLLinkElement and Setting Its href Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.href = "http://foo.bar/ ";
}
</html>
```

HTMMLinkElement.hreflang

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlLinkObj.hreflang`

Description

The hrefLang property of the HTMMLinkElement is the language code of the linked resource.

Example

Listing 11.203 illustrates the creation of an HTMMLinkElement using the HTMLDocument object element and then setting its href and hreflang properties.

Listing 11.203 *Creating an HTMMLinkElement and Setting Its href and hreflang Properties*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.href = "http://foo.bar/ ";
linkElement.hreflang = "us/english";
}
</script>
</html>
```

HTMMLinkElement.media

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlLinkObj.media`

Description

The media property of the HTMMLinkElement object is designed for use with one or more target media.

Example

Listing 11.204 illustrates the creation of a HTMMLinkElement and setting its media property.

Listing 11.204 *Setting an HTMLLinkElement's media Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.media = "image/jpg";
</script>
</html>
```

HTMLLinkElement.rel

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLink.rel

Description

The rel property of the HTMLLinkElement object is the forward link type.

Example

Listing 11.205 illustrates setting the rel property of the HTMLLinkElement object

Listing 11.205 *Setting the rel property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.href = "http://foo.bar/ ";
linkElement.rel = "http://foo.bar/link.html";
}
</script>
</html>
```

HTMLLinkElement.rev

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLinkObj.rev

Description

The rev property of the HTMLLinkElement object is the reverse link type.

Example

Listing 11.206 illustrates setting of the rev property of the HTMLLinkElement object.

Listing 11.206 Setting the rev property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.rev = "http://foo.bar/links.htm";
</script>
</html>
```

HTMLLinkElement.target

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLinkObj.target

Description

The target property of the HTMLLinkElement object is the frame to render the resource in.

Example

Listing 11.207 illustrates the creation of an HTMLLinkElement using the HTMLDocument object element and setting its target property.

Listing 11.207 Creating an HTMLLinkElement and Setting Its target Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
htmlDoc.write(linkElement.target);
}
// -->
</script>
</html>
```

HTMLLinkElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlLinkObj.type

Description

The type property of the HTMLLinkElement object is the advisory content type.

Example

Listing 11.208 illustrates the creation of an HTMLLinkElement using the HTMLDocument object element and then setting its type property.

Listing 11.208 *Creating an HTMLLinkElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("link");
linkElement.type = "text/html";
// -->
</script>
</html>
```

HTMLMapElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML element for a client-side image map. The HTMLMapElement object inherits all methods and properties of the HTMLInputElement object. Table 11.24 contains a list of properties for this object.

Table 11.24 *Properties of the HTMLMapElement Object*

Item	Description
areas	List of areas defined for this image map
name	The name of this image map

Example

Listing 11.209 illustrates the creation of an `HTMLMapElement` using the `HTMLBodyElement` object.

Listing 11.209 *Creating an* `HTMLMapElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");

// -->
</html>
```

HTMLMapElement.areas

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlMapObj.areas`

Description

The `areas` property of the `HTMLMapElement` object is the list of areas defined for the image map are contained in an `HTMLCollection`.

Example

Listing 11.210 illustrates the creation of an `HTMLMapElement` using the `HTMLBodyElement` object and then reading its `areas` property.

Listing 11.210 *Creating an* `HTMLMapElement` *and Reading Its* `areas` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
var areaCollection = mapObj.areas;

// -->
</html>
```

HTMLMapElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlMapObj.name`

Description

The name property of the HTMLMapElement object names the map (for use with UseMap).

Example

Listing 11.211 illustrates the creation of an HTMLMapElement using the HTMLBodyElement object and then setting its name property.

Listing 11.211 *Creating an HTMLMapElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var mapObj = bodyObj.createElement("map");
mapObj.name = "mymap";
// -->
</html>
```

HTMLMenuElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML menu element. The property for this object is as follows:

<i><u>Item</u></i>	<i><u>Description</u></i>
compact	Reduces line spacing between list items when true

Example

Listing 11.212 illustrates the creation of an HTMLMenuElement using the HTMLBodyElement.

Listing 11.212 *Creating an HTMLMenuElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var menuObj = bodyObj.createElement("menu");

// -->
</script>
</html>
```

HTMLMenuElement.compact

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlMenuObj.compact`

Description

The compact property of the HTMLMenuElement object indicates whether or not to reduce spacing between list items.

Example

Listing 11.213 illustrates the creation of an HTMLMenuElement using the HTMLBodyElement object and then setting its compact property.

Listing 11.213 *Creating an HTMLMenuElement and Setting Its compact Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var menuObj = bodyObj.CreateElement("menu");
menuObj.compact = true;
// -->
</script>
</html>
```

HTMLMetaElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core object represents DOM HTML object.

Description

This object represents generic meta-information about the document. HTMLMetaElement inherits all properties and methods from HTMLElement. Table 11.25 contains a list of properties for this object.

Table 11.25 *Properties of the HTMLMetaElement Object*

Item	Description
content	Associated information
httpEquiv	HTTP response header name
name	Meta information name
scheme	Form of content

Example

Listing 11.214 illustrates the creation of an HTMLMetaElement using the HTMLDocument object.

Listing 11.214 *Creating an* HTMLMetaElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var linkElement = htmlDoc.createElement("meta");
}
// -->
</script>
</html>
```

HTMLMetaElement.content

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlMetadataObj.content

Description

The content property of the HTMLMetadataElement object represents associated information contained in the metadata.

Example

Listing 11.215 illustrates the creation of an HTMLMetaElement using the HTMLDocument object element and then setting its content property.

Listing 11.215 *Creating an* HTMLMetaElement *and Setting Its* content Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var metaElement = htmlDoc.createElement("meta");
metaElement.content = "text/html";
}
// -->
</script>
</html>
```

HTMLMetaElement.httpEquiv

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlMetaDataObj.httpEquiv`

Description

The `httpEquiv` property of the `HTMLMetaElement` object is the HTTP response header name.

Example

Listing 11.216 illustrates the creation of an `HTMLMetaElement` using the `HTMLDocument` object element and then setting its `httpEquiv` property.

Listing 11.216 *Creating an HTMLMetaElement and Setting Its httpEquiv Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var metaElement = htmlDoc.createElement("meta");
metaElement.httpEquiv = "NO-CACHE";
}
// -->
</script>
</html>
```

HTMLMetaElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlMetaDataObj.name`

Description

The `name` property of the `HTMLMetaElement` object is the meta information name.

Example

Listing 11.217 illustrates the creation of an `HTMLMetaElement` using the `HTMLDocument` object element and then setting its `name` property.

Listing 11.217 *Creating an HTMLMetaElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var metaElement = htmlDoc.createElement("meta");
metaElement.name = "keywords";
}
// -->
</script>
</html>
```

HTMLMetaElement.scheme

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlMetaDataObj.scheme

Description

The scheme property of the HTMLMetaElement object is the select form of content.

Example

Listing 11.218 illustrates the creation of an HTMLMetaElement using the HTMLDocument object element and then setting its scheme property.

Listing 11.218 *Creating an HTMLMetaElement and Setting Its scheme Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var metaElement = htmlDoc.createElement("meta");
metaElement.scheme = "foo";
}
// -->
</script>
</html>
```

HTMLModElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents a notice of modification to part of a document. The HTMLModElement object inherits all methods and properties from HTMLElement. Table 11.26 contains a list of properties for this object.

Table 11.26 *Properties of the HTMLModElement Object*

Item	Description
<code>cite</code>	URI for the document that indicates the reason for the change
<code>dateTime</code>	The time that the change occurred

Example

Listing 11.219 illustrates the creation of an HTMLModElement using the HTMLBodyElement object.

Listing 11.219 *Creating an HTMLModElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var modObj = bodyObj.CreateElement("mod");

// -->
</script>
</html>
```

HTMLModElement.cite

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlModObj.cite`

Description

The `cite` property of the HTMLModElement object is a URI designating a document that describes the reason for the change.

Example

Listing 11.220 illustrates the creation of an `HTMLModElement` using the `HTMLBodyElement` object element and then setting its `cite` property.

Listing 11.220 *Creating an `HTMLModElement` and Setting Its `cite` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var modObj = bodyObj.createElement("mod");
modObj.cite = "http://foo.bar/reference.html";
// -->
</script>
</html>
```

HTMLModElement.dateTime

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlModObj.dateTime`

Description

The `dateTime` property of the `HTMLModElement` is the date and time of the change.

Example

Listing 11.221 illustrates the creation of an `HTMLModElement` using the `HTMLBodyElement` object and then setting its `dateTime` property.

Listing 11.221 *Creating an `HTMLModElement` and Setting Its `dateTime` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var modObj = bodyObj.CreateElement("mod");
modObj.cite = "http://foo.bar/reference.html";
modObj.dateTime = "2/14/2001";
// -->
</script>
</html>
```

HTMLObjectElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML object element. The HTMLObjectElement object inherits all methods and properties from the HTMLElement object. Table 11.27 contains a list of properties for this object.

Table 11.27 Properties of the HTMLObjectElement *Object*

Item	Description
align	The horizontal alignment for this object element
archive	The archive file associated with this object element
border	The border width surrounding this object
code	The <code>Applet</code> class file for this object element
codeBase	Base URI for data and code
codeType	Content for data
contentDocument	The document contained by this object
data	URI specifying this object element's data
declare	Declares but doesn't instantiate this object
form	The form in which this object element resides
height	The height of this object element
hspace	Space to the left and right of this object element
name	The name of this object element
standby	Message to render while object is loading
tabIndex	Number indicating the tab order of this object element
type	Content type of data
useMap	Indicates whether to use a client side image map
vspace	Space above and below the object element
width	The width of this object element

Example

Listing 11.222 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then reading its form property.

Listing 11.222 Creating an HTMLObjectElement and Reading Its form Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

```
var objectObj = bodyObj.createElement("object");
if(objectObj.Form.Name == "emailForm")
    processEmailForm(objectObj.form);
// -->
</script>
</html>
```

HTMLObjectElement.align

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

htmlObjectObj.align

Description

The align property of the HTMLObjectElement object aligns this object (vertically or horizontally) with respect to its surrounding text.

Example

Listing 11.223 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its align property.

Listing 11.223 *Creating an HTMLObjectElement and Setting Its align Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.align = "top";
// -->
</script>
</html>
```

HTMLObjectElement.archive

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

htmlObjectObj.archive

Description

The archive property of the HTMLObjectElement object is the space-separated list of archives.

Example

Listing 11.224 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `archive` property.

Listing 11.224 *Creating an `HTMLObjectElement` and Setting Its `archive` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.archive = "foo.jar";
// -->
</script>
</html>
```

HTMLObjectElement.border

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.border`

Description

The `border` property of the `HTMLObjectElement` object is the width of border around the object.

Example

Listing 11.225 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `border` property.

Listing 11.225 *Creating an `HTMLObjectElement` and Setting Its `border` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.border = 0;
// -->
</script>
</html>
```


HTMLObjectElement.code

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlObjectObj.code

Description

The code property of the HTMLObjectElement object is an applet class file. See the code attribute for HTMLAppletElement.

Example

Listing 11.226 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its code property.

Listing 11.226 *Creating an HTMLObjectElement and Setting Its code Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.code = "classId";
// -->
</script>
</html>
```

HTMLObjectElement.codeBase

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlObjectObj.codeBase

Description

The codeBase property of the HTMLObjectElement object is the base URI for classid, data, and archive attributes.

Example

Listing 11.227 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its codeBase property.

Listing 11.227 *Creating an `HTMLObjectElement` and Setting Its `codeBase` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.codeBase = "http://foo.bar/scripts/";
// -->
</script>
</html>
```

HTMLObjectElement.codeType

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.codeType`

Description

The `codeType` property of the `HTMLObjectElement` is the content type for data downloaded via `classid` attribute.

Example

Listing 11.228 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `codeType` property.

Listing 11.228 *Creating an `HTMLObjectElement` and Setting Its `codeType` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.codeType = "x-application/flash";
// -->
</script>
</html>
```

HTMLObjectElement.contentDocument

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.contentDocument`

Description

The `contentDocument` property of the `HTMLObjectElement` is the document this object contains, if there is any and it is available, or `Null` otherwise.

Example

Listing 11.229 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then reading its `contentDocument` property.

Listing 11.229 *Creating an `HTMLObjectElement` and Setting Its `contentDocument` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = objectObj.contentDocument;
// -->
</script>
</html>
```

HTMLObjectElement.data

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.data`

Description

The `data` property of the `HTMLObjectElement` object is a URI specifying the location of the object's data.

Example

Listing 11.230 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object.

Listing 11.230 *Creating an `HTMLObjectElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");

// -->
</script>
</html>
```

HTMLObjectElement.declare

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.declare`

Description

The declare property of the HTMLObjectElement object declares (for future reference), but does not instantiate, this object.

Example

Listing 11.231 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its declare property.

Listing 11.231 *Creating an HTMLObjectElement and Setting Its declare Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.declare = true;
// -->
</script>
</html>
```

HTMLObjectElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.form`

Description

The form property of the HTMLObjectElement object returns the FORM element containing this control. Returns Null if this control isn't within the context of a form.

Example

Listing 11.232 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object element and then reading its form property.

Listing 11.232 *Creating an* `HTMLObjectElement` *and Reading Its* `form` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
if(objectObj.form.name == "emailForm")
    processEmailForm(objectObj.form);
// -->
</script>
</html>
```

HTMLObjectElement.height

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.height`

Description

The height property of the `HTMLObjectElement` object overrides the height.

Example

Listing 11.233 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its height property.

Listing 11.233 *Creating an* `HTMLObjectElement` *and Setting Its* `height` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.height = 23;
// -->
</script>
</html>
```

HTMLObjectElement.hspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.hspace`

Description

The `hspace` property of the `HTMLObjectElement` is the horizontal space to the left and right of this image, applet, or object.

Example

Listing 11.234 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `hspace` property.

Listing 11.234 *Creating an `HTMLObjectElement` and Setting Its `hspace` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.hspace = 1;
// -->
</script>
</html>
```

HTMLObjectElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.name`

Description

The `name` property of the `HTMLObjectElement` object is the form control or object name when submitted with a form.

Example

Listing 11.235 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `name` property.

Listing 11.235 *Creating an `HTMLObjectElement` and Setting Its `name` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.name = "myobject";
// -->
</script>
</html>
```

HTMLObjectElement.standby

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.standby`

Description

The `standby` property of the `HTMLObjectElement` object is the message to render while loading the object.

Example

Listing 11.236 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `standby` property.

Listing 11.236 *Creating an `HTMLObjectElement` and Setting Its `standby` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.standby = "Please wait while loading object.";
// -->
</script>
</html>
```

HTMLObjectElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.tabIndex`

Description

The `tabIndex` property of the `HTMLObjectElement` is the index that represents the element's position in the tabbing order.

Example

Listing 11.237 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `tabIndex` property.

Listing 11.237 *Creating an `HTMLObjectElement` and Setting Its `tabIndex` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.tabIndex = 3;
// -->
</script>
</html>
```

HTMLObjectElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.type`

Description

The `type` property of the `HTMLObjectElement` object is the content type for data downloaded via data attribute.

Example

Listing 11.238 illustrates the creation of an `HTMLObjectElement` using the `HTMLBodyElement` object and then setting its `type` property.

Listing 11.238 *Creating an `HTMLObjectElement` and Setting Its `type` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.type = "applicationType";
// -->
</script>
</html>
```

HTMLObjectElement.useMap

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlObjectObj.useMap`

Description

The useMap property of the HTMLObjectElement object indicates whether or not to use a client-side image map.

Example

Listing 11.239 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its useMap property.

Listing 11.239 *Creating an HTMLObjectElement and Setting Its useMap Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.useMap = true;
// -->
</script>
</html>
```

HTMLObjectElement.vspace

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlObjectObj.vspace

Description

The vspace property of the HTMLObjectElement object is the vertical space above and below this image, applet, or object.

Example

Listing 11.240 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its vspace property.

Listing 11.240 *Creating an HTMLObjectElement and Setting Its vspace Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.vspace = 2;
// -->
</script>
</html>
```

HTMLObjectElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlObjectObj.width

Description

The width property of the HTMLObjectElement object overrides the width of the object.

Example

Listing 11.241 illustrates the creation of an HTMLObjectElement using the HTMLBodyElement object and then setting its width property.

Listing 11.241 *Creating an HTMLObjectElement and Setting Its width Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var objectObj = bodyObj.createElement("object");
objectObj.width = 20;
// -->
</script>
</html>
```

HTMLListElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML ordered list element. The HTMLListElement object inherits all methods and properties of the HTML element object. Table 11.28 contains a list of properties for this object.

Table 11.28 *Properties of the HTMLListElement Object*

Item	Description
compact	Reduces the spacing between list items if true
start	The starting index of the first list item
type	The bullet type for this ordered list element

Example

Listing 11.242 illustrates the creation of an `HTMLListElement` using the `HTMLBodyElement` object.

Listing 11.242 *Creating an* `HTMLListElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElement("ol");
// -->
</script>
</html>
```

`HTMLListElement.compact`

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

`htmlListObj.compact`

Description

The `compact` property of the `HTMLListElement` object reduces spacing between list items.

Example

Listing 11.243 illustrates the creation of an `HTMLListElement` using the `HTMLBodyElement` object and then setting its `compact` property.

Listing 11.243 *Creating an* `HTMLListElement` *and Setting Its* `compact` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElement("ol");
olObj.compact = true;
// -->
</script>
</html>
```

`HTMLListElement.start`

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

`htmlListObj.start`

Description

The start property of the HTMLListElement object is the starting sequence number.

Example

Listing 11.244 illustrates the creation of an HTMLListElement using the HTMLBodyElement object and then setting its start property.

Listing 11.244 *Creating an HTMLListElement and Setting Its start Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElement("ol");
olObj.start = 1;
// -->
</script>
</html>
```

HTMLListElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlListObj.type

Description

The type property of the HTMLListElement object indicates its numbering style.

Example

Listing 11.245 illustrates the creation of an HTMLListElement using the HTMLBodyElement object and then setting its type property.

Listing 11.245 *Creating an HTMLListElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var olObj = bodyObj.CreateElement("ol");
olObj.type = "ascending";
// -->
</script>
</html>
```

HTMLOptGroupElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

Group options together in logical subdivisions. This object inherits all methods and properties from `HTMLElement`. Table 11.29 contains a list of properties for this object.

Table 11.29 *Properties of the* `HTMLOptGroupElement` *Object*

Item	Description
<code>disabled</code>	Indicates whether this element is disabled
<code>label</code>	The label for this option group

Example

Listing 11.246 illustrates the creation of an `HTMLOptGroupElement` using the `HTMLSelectElement` object.

Listing 11.246 *Creating an* `HTMLOptGroupElement` *Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
// -->
</script>
</html>
```

HTMLOptGroupElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlOptGroupObj.disabled`

Description

The `disabled` property of the `HTMLOptGroupElement` is used to disable the option group of elements.

Example

Listing 11.247 illustrates the creation of an `HTMLOptGroupElement` using the `HTMLSelectElement` object and then setting its disabled property.

Listing 11.247 Creating an `HTMLOptGroupElement` and Setting Its disabled Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
optionGrp.disabled = true;
// -->
</script>
</html>
```

HTMLOptGroupElement.label

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlOptGroupObj.label`

Description

The `label` property of the `HTMLOptGroupElement` assigns a label to this option group.

Example

Listing 11.248 illustrates the creation of an `HTMLOptGroupElement` using the `HTMLSelectElement` object and then setting its `label` property.

Listing 11.248 Creating an `HTMLOptGroupElement` and Setting Its label Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
```

```
optionGrp.label = "Colors";  
// -->  
</script>  
</html>
```

HTMLOptionElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML option element that resides in a select element. This object inherits all methods and properties from `HTMLElement`. Table 11.30 contains a list of properties for this object.

Table 11.30 *Properties of the HTMLOptionElement Object*

Item	Description
<code>defaultSelected</code>	Indicates whether this option is selected by default
<code>disabled</code>	Indicates whether this option is disabled
<code>form</code>	The form where this element resides
<code>index</code>	The index number of this option
<code>label</code>	Label for this option when used as a hierarchical menu
<code>selected</code>	Indicates whether this option is currently selected
<code>text</code>	The text of this option
<code>value</code>	The value of this option

Example

Listing 11.249 illustrates the creation of an `HTMLOptionElement` using the `HTMLOptGroupElement` object.

Listing 11.249 *Creating an HTMLOptionElement*

```
<html>  
<script language="JavaScript" type="text/javascript">  
!--  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var bodyObj = htmlDoc.createElement("body");  
var formObj = bodyObj.createElement("form");  
var selectElement = formObj.createElement("select");  
var optionGrp = selectElement.createElement("options");  
var optionElement = optionGrp.createElement("option");
```

Listing 11.249 Continued

```
// -->
</script>
</html>
```

HTMLOptionElement.defaultSelected

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.defaultSelected

Description

The defaultSelected property of the HTMLOptionElement object represents the value of the HTML selected attribute. The value of this attribute doesn't change if the state of the corresponding form control, in an interactive user agent, changes. Changing defaultSelected, however, resets the state of the form control.

Example

Listing 11.250 illustrates the creation of an HTMLOptionElement using the HTMLOptGroupElement object and then setting its defaultSelected property.

Listing 11.250 Creating an HTMLOptionElement and Setting Its defaultSelected Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
optionElement.defaultSelected = true;
// -->
</script>
</html>
```

HTMLOptionElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.disabled

Description

The disabled property of the HTMLOptionElement object is used to disable the option instance.

Example

Listing 11.251 illustrates the creation of an HTMLOptionElement using the HTMLSelectElement object and then setting its disabled property.

Listing 11.251 *Creating an HTMLOptionElement and Setting Its disabled Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
optionElement.disabled = true;
// -->
</script>
</html>
```

HTMLOptionElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.form

Description

The form property of the HTMLOptionElement object returns the FORM element containing this control. Returns Null if this control isn't within the context of a form.

Example

Listing 11.252 illustrates the creation of an HTMLOptionElement using the HTMLSelectElement object and then reading its form property.

Listing 11.252 *Creating an HTMLOptionElement and Reading Its form Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

Listing 11.252 Continued

```
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
if(optionElement.form == "myForm")
    processOption(optionElement);
// -->
</script>
</html>
```

HTMLOptionElement.index

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.index

Description

The index property of the HTMLOptionElement object is the index of this option in its parent SELECT, starting from 0.

Example

Listing 11.253 illustrates the creation of an HTMLOptionElement using the HTMLSelectElement object and then reading its index property.

Listing 11.253 Creating an HTMLOptionElement and Reading Its index Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
if(optionElement.index == 0)
    processFirstOption(optionElement);
// -->
</script>
</html>
```

HTMLOptionElement.label

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.label

Description

The label property of the HTMLOptionElement object is the option label for use in hierarchical menus.

Example

Listing 11.254 illustrates the creation of an HTMLOptionElement using the HTMLSelectElement object and then setting its label property.

Listing 11.254 *Creating an HTMLOptionElement and Setting Its label Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
optionElement.label = "Green";
// -->
</script>
</html>
```

HTMLOptionElement.selected

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.selected

Description

The selected property of the HTMLOptionElement object represents the current state of the corresponding form control, in an interactive user agent. Changing this attribute changes the state of the form control, but does not change the value of the HTML selected attribute of the element.

Example

Listing 11.255 illustrates the creation of an `HTMLOptionElement` using the `HTMLSelectElement` object and then setting its selected property.

Listing 11.255 *Creating an `HTMLOptionElement` and Setting Its selected Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
optionElement.selected = "true";
// -->
</script>
</html>
```

HTMLOptionElement.text

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlOptionObj.text`

Description

The text property of the `HTMLOptionElement` object is the text contained within the option element.

Example

Listing 11.256 illustrates the creation of an `HTMLOptionElement` using the `HTMLSelectElement` object and then setting its text property.

Listing 11.256 *Creating an `HTMLOptionElement` and Setting Its text Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
```

```
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
var optionElement.text = "Blue";
// -->
</script>
</html>
```

HTMLOptionElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlOptionObj.value

Description

The value property of the HTMLOptionElement object is the current form control value.

Example

Listing 11.257 illustrates the creation of an HTMLOptionElement using the HTMLSelectElement object and then setting its value property.

Listing 11.257 *Creating an HTMLOptionElement and Setting Its value Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
var optionGrp = selectElement.createElement("options");
var optionElement = optionGrp.createElement("option");
optionElement.value = "g";
// -->
</script>
</html>
```

HTMLParagraphElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML paragraph element. The HTMLParagraph element inherits all methods and properties of the HTMLElement object. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
align	The horizontal text alignment for this paragraph element

Example

Listing 11.258 illustrates the creation of an HTMLParagraphElement using the HTMLBodyElement object and then setting its align property.

Listing 11.258 Creating an HTMLParagraph and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var paragraphObj = bodyObj.CreateElement("p");
paragraphObj.align = "top";
// -->
</script>
</html>
```

HTMLParagraphElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlParagraphObj.align

Description

The align property of the HTMLParagraphElement object is the horizontal text alignment.

Example

Listing 11.259 illustrates the creation of an HTMLParagraphElement using the HTMLBodyElement object and then setting its align property.

Listing 11.259 Creating an HTMLParagraphElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var paragraphObj = bodyObj.CreateElement("p");
paragraphObj.align = "bottom";
```

```
// -->
</script>
</html>
```

HTMLParamElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML parameter element. The HTMLParamElement object inherits all methods and properties of the HTMLInputElement object. Table 11.31 contains a list of properties for this object.

Table 11.31 *Properties of the HTMLParamElement Object*

Item	Description
name	The name of this parameter element
type	The data type of this parameter element
value	The value of this runtime parameter element
valueType	Information about the meaning of the value property

Example

Listing 11.260 illustrates the creation of an HTMLParamElement using the HTMLMetaElement object.

Listing 11.260 *Creating an HTMLParamElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var metaObj = bodyObj.createElement("meta");
var paramObj = metaObj.createElement("param");

// -->
</script>
</html>
```

HTMLParamElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlParamObj.name

Description

The name property of the HTMLParamElement object is the name of a runtime parameter.

Example

Listing 11.261 illustrates the creation of an HTMLParamElement using the HTMLMetaElement object and then setting its name.

Listing 11.261 *Creating an HTMLParamElement and Setting Its name*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var metaObj = bodyObj.createElement("meta");
var paramObj = metaObj.createElement("param");
paramObj.name = "keywords"

// -->
</script>
</html>
```

HTMLParamElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlParamObj.type

Description

The type property of the HTMLParamElement is the content type for the value attribute when valuetype has the value "ref".

Example

Listing 11.262 illustrates the creation of an HTMLParamElement using the HTMLMetaElement object and then setting its name, value, and type properties.

Listing 11.262 *Creating an HTMLParamElement and Setting Its name, value, and type Properties*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var metaObj = bodyObj.createElement("meta");
var paramObj = metaObj.createElement("param");
paramObj.name = "keywords"
paramObj.value = "horses,cows,sheep";
paramObj.type = "animals";
```



```
// -->
</script>
</html>
```

HTMLParamElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlParamObj.value

Description

The value property of the HTMLParamElement object is the value of a runtime parameter.

Example

Listing 11.263 illustrates the creation of an HTMLParamElement using the HTMLMetaElement object and then setting its value property.

Listing 11.263 *Creating an HTMLParamElement and Setting Its value Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var metaObj = bodyObj.createElement("meta");
var paramObj = metaObj.createElement("param");

paramObj.value = "horses,cows,sheep";
// -->
</script>
</html>
```

HTMLParamElement.valueType

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlParamObj.valueType

Description

The valueType property of the HTMLParamElement object is information about the meaning of the value attribute value.

Example

Listing 11.264 illustrates the creation of an `HTMLParamElement` using the `HTMLMetaElement` object element and then setting its `valueType` and `name` properties.

Listing 11.264 Creating an `HTMLParamElement` and Setting Its `valueType` and `name` Properties

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var metaObj = bodyObj.createElement("meta");
var paramObj = metaObj.createElement("param");
paramObj.name = "keywords";
paramObj.valueType = "ref";
// -->
</script>
</html>
```

HTMLPreElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML pre-formatted text element. The `HTMLPreElement` inherits all properties and methods of `HTMLElement`. The property for this object is as follows:

<u><i>Item</i></u>	<u><i>Description</i></u>
<code>width</code>	The fixed width for the contents of the PRE element

Example

Listing 11.265 illustrates the creation of an `HTMLPreElement` using the `HTMLBodyElement` object.

Listing 11.265 Creating an `HTMLPreElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var preObj = bodyObj.createElement("pre");

// -->
</script>
</html>
```

HTMLPreElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlPreObj.width

Description

The width property of the HTMLPreElement object is the fixed width for content.

Example

Listing 11.266 illustrates the creation of an HTMLPreElement using the HTMLBodyElement object and then setting its width property.

Listing 11.266 *Creating an HTMLPreElement and Setting Its width Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var preObj = bodyObj.CreateElement("pre");
preObj.width = 25;
// -->
</script>
</html>
```

HTMLQuoteElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents both HTML Q and BLOCKQUOTE elements. The HTMLQuoteElement inherits all properties and methods from HTMLInputElement. The property for this object is as follows:

<u>Item</u>	<u>Description</u>
cite	URI designating source message

Example

Listing 11.267 illustrates the creation of an HTMLQuoteElement using the HTMLBodyElement object.

Listing 11.267 *Creating an* HTMLQuoteElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var quoteObj = bodyObj.CreateElement("quote");

// -->
</script>
</html>
```

HTMLQuoteElement.cite

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlQuoteObj.cite

Description

The *cite* property of the HTMLQuoteElement object is a URI designating a source document or message.

Example

Listing 11.268 illustrates the creation of an HTMLQuoteElement using the HTMLBodyElement object and then setting its *cite* property.

Listing 11.268 *Creating an* HTMLQuoteElement *and Setting Its* cite Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var quoteObj = bodyObj.CreateElement("quote");
quoteObj.cite = "http://foo.bar/reference.html";
// -->
</script>
</html>
```

HTMLScriptElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML script element. `HTMLScriptElement` inherits all properties and methods from the `HTMLElement` object. Table 11.32 contains a list of properties for this object.

Table 11.32 *Properties of the `HTMLScriptElement` Object*

Item	Description
<code>charset</code>	Character encoding for script element
<code>defer</code>	Indicates whether the user agent can defer execution of the script
<code>event</code>	Reserved for future use
<code>htmlFor</code>	Reserved for future use
<code>src</code>	URI designating external script
<code>text</code>	Script content for this script element
<code>type</code>	Content type of the script language

Example

Listing 11.269 illustrates the creation of an `HTMLScriptElement` using the `HTMLBodyElement` object and then setting its `text` property.

Listing 11.269 *Creating an `HTMLScriptElement` and Setting Its `text` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");
scriptObj.text = "some script text";

// -->
</script>
</html>
```

HTMLScriptElement.charset

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlScriptObj.charset`

Description

The `charset` property of the `HTMLScriptElement` object is the character encoding of the linked resource.

Example

Listing 11.270 illustrates the creation of an `HTMLScriptElement` using the `HTMLBodyElement` object and then setting its `charset` property.

Listing 11.270 *Creating an `HTMLScriptElement` and Setting `charset` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");
scriptObj.charset = "us/english";

// -->
</script>
</html>
```

HTMLScriptElement.defer

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlScriptObj.defer`

Description

The `defer` property of the `HTMLScriptElement` object indicates that the user agent can defer processing of the script.

Example

Listing 11.271 illustrates the creation of an `HTMLScriptElement` using the `HTMLBodyElement` object and then setting its `defer` property.

Listing 11.271 *Creating an `HTMLScriptElement` and Setting Its `defer` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");
scriptObj.defer = true;

// -->
</script>
</html>
```

HTMLScriptElement.event

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`htmlScript.event`

Description

The event property of the HTMLScriptElement object is reserved for future use.

HTMLScriptElement.htmlFor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlScriptObj.htmlFor`

Description

The htmlFor property of the HTMLScriptElement object is reserved for future use.

HTMLScriptElement.src

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlScriptObj.src`

Description

The src property of the HTMLScriptElement object is a URI designating an external script.

Example

Listing 11.272 illustrates the creation of an HTMLScriptElement using the HTMLBodyElement object.

Listing 11.272 *Creating an* HTMLScriptElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");

// -->
</script>
</html>
```

HTMLScriptElement.text

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlScriptObj.text

Description

The text property of the HTMLScriptElement object is the script content of the element.

Example

Listing 11.273 illustrates the creation of an HTMLScriptElement using the HTMLBodyElement object element and then setting its text property.

Listing 11.273 *Creating an HTMLScriptElement and Setting Its text Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");
scriptObj.text = "some script text";

// -->
</script>
</html>
```

HTMLScriptElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlScriptObj.type

Description

The type property of the HTMLScriptElement is the content type of the script language.

Example

Listing 11.274 illustrates the creation of an HTMLScriptElement using the HTMLBodyElement object and then setting its type property.

Listing 11.274 *Creating an `HTMLScriptElement` and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var scriptObj = bodyObj.createElement("script");
scriptObj.type = "text/html";

// -->
</script>
</html>
```

HTMLSelectElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

The select element allows the selection of an option. The contained options can be directly accessed through the select element as a collection. This object inherits all methods and properties from `HTMLElement`. Table 11.33 contains a list of properties and methods for this object.

Table 11.33 *Contents of the `HTMLSelectElement` Object*

Type	Item	Description
Property	<code>disabled</code>	Indicates if this select element is disabled.
	<code>form</code>	The form in which this select element exists.
	<code>length</code>	Number of options in this selection element.
	<code>multiple</code>	Indicates if multiple selections are allowed.
	<code>name</code>	The name of this selection element.
	<code>options</code>	List of all options present in this select element.
	<code>selectedIndex</code>	Index of the selected option.
	<code>size</code>	Number of visible rows for this selection element.
	<code>tabIndex</code>	This element's position in the tabbing order.
	<code>type</code>	The type of this select element.
Method	<code>value</code>	The value of the currently selected option.
	<code>add()</code>	Adds an option to the selection element.
	<code>blur()</code>	Removes keyboard focus from this element.
	<code>focus()</code>	Gives keyboard focus to this element.
	<code>remove()</code>	Remove an element from the Options list.

Example

Listing 11.275 illustrates the creation of an `HTMLSelectElement` using the `HTMLFormElement` object.

Listing 11.275 *Creating an `HTMLSelectElement` Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectElement = formObj.createElement("select");
// -->
</script>
</html>
```

HTMLSelectElement.add()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.add(element, before)`

Description

The `add()` method of the `HTMLSelectElement` object adds a new element to the collection of `OPTION` elements for this `SELECT`. This method is equivalent to the `AppendChild()` method of the `Node` object if the *before* parameter is `Null`. It is equivalent to the `InsertBefore()` method on the parent of *before* in all other cases.

Example

Listing 11.276 illustrates the creation of a `HTMLSelectElement` and invoking its `add()` method.

Listing 11.276 *Creating an `HTMLSelectElement` and Invoking `add()`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
var optionObj = htmlDoc.createElement("option");
selectObj.add(optionObj, null);
```

```
// -->  
<script>  
</html>
```

HTMLSelectElement.blur()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.blur()`

Description

The `blur()` method of the `HTMLSelectElement` object removes keyboard focus from this element.

Example

Listing 11.277 illustrates the creation of a `HTMLSelectElement` and invoking its `blur()` method.

Listing 11.277 *Creating an HTMLSelectElement and Invoking blur()*

```
<html>  
<script language="JavaScript" type="text/javascript">  
!--  
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");  
htmlDoc.open();  
var bodyObj = htmlDoc.createElement("body");  
var formObj = bodyObj.createElement("form");  
var selectObj = formObj.createElement("select");  
selectObj.blur();  
// -->  
</script>  
</html>
```

HTMLSelectElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.disabled`

Description

The `disabled` property of the `HTMLSelectElement` object is used to disable the select element instance.

Example

Listing 11.278 illustrates the creation of an `HTMLSelectElement` using an `HTMLFormElement` object and setting its disabled property.

Listing 11.278 *Creating an `HTMLSelectElement` and Setting disabled*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.disabled = true;
}
// -->
</script>
</html>
```

HTMLSelectElement.focus()

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

```
htmlSelectObj.focus()
```

Description

The `focus()` method of the `HTMLSelectElement` object gives keyboard focus to this element.

Example

Listing 11.279 illustrates the creation of a `HTMLSelectElement` and invoking its `focus()` method.

Listing 11.279 *Creating an `HTMLSelectElement` and Invoking `focus()`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.focus()
// -->
</script>
</html>
```

HTMLSelectElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.form`

Description

The `form` property of the `HTMLSelectElement` object returns the `FORM` element containing this control. Returns `Null` if this control isn't within the context of a form.

Example

Listing 11.280 illustrates the reading an `HTMLSelectElement`'s `form` property.

Listing 11.280 *Reading the `form` Property of `HTMLSelectElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
function submitFormFromSelect(selectObj) {
    var formObj = selectObj.form;
    formObj.submit();
}
// -->
</script>
</html>
```

HTMLSelectElement.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.length`

Description

The `length` property of the `HTMLSelectElement` object is the number of options in this `SELECT`.

Example

Listing 11.281 illustrates the creation of a `HTMLSelectObject` and reading its `length` property.

Listing 11.281 *Reading the `length` Property of `HTMLSelectObject`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

Listing 11.281 Continued

```
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
var optionList = selectObj.options;
var i = 0;
while( i < selectObj.length) {
    processOption(optionList.index(i));
    i++;
}
// -->
</script>
</html>
```

HTMLSelectElement.multiple

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.multiple

Description

The `multiple` property of the `HTMLSelectElement` object indicates whether or not multiple `OPTION` elements can be selected in this `SELECT`.

Example

Listing 11.282 illustrates the creation of a `HTMLSelectElement` and setting its `multiple` property.

Listing 11.282 Setting the *multiple* Property of HTMLSelectElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.multiple = true;
}
// -->
</script>
</html>
```

HTMLSelectElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.name

Description

The name property of the HTMLSelectElement object is the form control or object name when submitted with a form.

Example

Listing 11.283 illustrates the creation of a HTMLSelectElement and setting its name property.

Listing 11.283 *Setting the name Property of* HTMLSelectElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.name = "Color Selection";
// -->
</script>
</html>
```

HTMLSelectElement.options

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.options

Description

The options property of the HTMLSelectElement object is the collection of OPTION elements contained by this element.

Example

Listing 11.284 illustrates the creation of a HTMLSelectElement and iterating through its options property.

Listing 11.284 Iterating Through an `HTMLSelectElement`'s options Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
var optionList = selectObj.options;
var i = 0;
while( i < selectObj.length) {
    processOption(optionList.index(i));
    i++;
}
// -->
</script>
</html>
```

HTMLSelectElement.remove()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.remove(index)

Description

The `remove()` method of the `HTMLSelectElement` object removes an element from the collection of `OPTION` elements for this `SELECT`. Does nothing if no element has the given *index*.

Example

Listing 11.285 illustrates the creation of a `HTMLSelectElement` and invoking its `remove()` method.

Listing 11.285 Invoking the `remove()` Method of `HTMLSelectElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.remove(selectObj.selectedIndex);
```



```
// -->
</script>
</html>
```

HTMLSelectElement.selectedIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.selectedIndex

Description

The selectedIndex property of the HTMLSelectElement object is the ordinal index of the selected option, starting from 0. The value -1 is returned if no element is selected. If multiple options are selected, the index of the first selected option is returned.

Example

Listing 11.286 illustrates the creation of an HTMLSelectElement using the HTMLFormElement object and then setting its selectedIndex property.

Listing 11.286 *Creating an HTMLSelectElement and Setting Its selectedIndex Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
var optionObj = htmlDoc.createElement("option");
selectObj.Add(optionObj);
selectObj.selectedIndex = 0;
// -->
</script>
</html>
```

HTMLSelectElement.size

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.size

Description

The size property of the `HTMLSelectElement` object is the number of visible rows.

Example

Listing 11.287 illustrates the creation of an `HTMLSelectElement` and setting its size property.

Listing 11.287 *Setting the size Property of* `HTMLSelectElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.size = 20;
}
// -->
</script>
</html>
```

HTMLSelectElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlSelectObj.tabIndex`

Description

The `tabIndex` property of the `HTMLSelectElement` is the index that represents the element's position in the tabbing order.

Example

Listing 11.288 illustrates the creation of an `HTMLSelectElement` and setting its `tabIndex` property.

Listing 11.288 *Setting the tabIndex Property of* `HTMLSelectElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
```

```
var selectObj = formObj.createElement("select");
selectObj.tabIndex = 4;
// -->
</script>
</html>
```

HTMLSelectElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.type

Description

The type property of the HTMLSelectElement object is the type of this form control. This is the string "select-multiple" when the multiple attribute is True and the string "select-one" when False.

Example

Listing 11.289 illustrates the creation of an HTMLSelectElement using the HTMLFormElement object and then setting its type property.

Listing 11.289 *Creating an HTMLSelectElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.type = "select-multiple";
// -->
</script>
</html>
```

HTMLSelectElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlSelectObj.value

Description

The value property of the HTMLSelectElement object is the current form control value.

Example

Listing 11.290 illustrates the creation of an HTMLSelectElement using the HTMLFormElement object and then setting its value property.

Listing 11.290 *Creating an HTMLSelectElement and Setting Its value Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var bodyObj = htmlDoc.createElement("body");
var formObj = bodyObj.createElement("form");
var selectObj = formObj.createElement("select");
selectObj.value = "selection text";
// -->
</script>
</html>
```

HTMLStyleElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

Represents style information contained in a style sheet. HTMLStyleElement inherits all methods and properties from HTMLInputElement. Table 11.34 contains a list of properties for this object.

Table 11.34 *Properties of the HTMLStyleElement Object*

Item	Description
disabled	Indicates whether this style is disabled
media	The target media for this style
type	Content type of the style sheet language

Example

Listing 11.291 illustrates the creation of an HTMLStyleElement using the HTMLDocument.

Listing 11.291 *Creating an* HTMLStyleElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var styleObj = htmlDoc.createElement("style");

// -->
</script>
</html>
```

HTMLStyleElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

styleInfoObj.disabled

Description

The disabled property of the HTMLStyleElement enables/disables the style sheet.

Example

Listing 11.292 illustrates the creation of an HTMLStyleObject using the HTMLDocument object element and then disabling it.

Listing 11.292 *Creating an* HTMLStyleElement *and Setting* disabled Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var styleObj = htmlDoc.createElement("style");
styleObj.disabled = true;
// -->
</script>
</html>
```

HTMLStyleElement.media

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

styleInfoObj.media

Description

The `media` property of the `HTMLStyleElement` object is designed for use with one or more target media.

Example

Listing 11.293 illustrates the creation of an `HTMLStyleObject` using the `HTMLDocument` object element and then setting its `media` property.

Listing 11.293 Creating an `HTMLStyleObject` and Setting Its `media` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var styleObj = htmlDoc.createElement("style");
styleObj.media = "jpg";
// -->
</script>
</html>
```

HTMLStyleElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`styleInfoObj.type`

Description

The `type` property of the `HTMLStyleElement` object is the content type of the style sheet language.

Example

Listing 11.294 illustrates the creation of an `HTMLStyleObject` using the `HTMLDocument` object element and then setting its `type` property.

Listing 11.294 Creating an `HTMLStyleObject` and Setting Its `type` Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var styleObj = htmlDoc.createElement("style");
styleObj.type = "text/html";
```

```
// -->
</script>
</html>
```

HTMLTableCaptionElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML table caption element. `HTMLTableCaptionElement` inherits all properties and methods from `HTMLElement`. The property for this object is as follows:

<i>Item</i>	<i>Description</i>
<code>align</code>	Aligns caption with respect to the table it resides in

Example

Listing 11.295 illustrates the creation of an `HTMLTableCaptionElement` using the `HTMLTableElement` object.

Listing 11.295 *Creating an* `HTMLTableCaptionElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var captionObj = tableObj.createCaption();

// -->
</script>
</html>
```

HTMLTableCaptionElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCaptionObj.align`

Description

The `align` property of the `HTMLTableCaptionElement` is the caption alignment with respect to the table.

Example

Listing 11.296 illustrates the creation of an `HTMLTableCaptionElement` using the `HTMLTableElement` object and then setting its `align` property.

Listing 11.296 *Creating an `HTMLTableCaptionElement` and Setting `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var captionObj = tableObj.createCaption();
captionObj.align = "top";
// -->
</script>
</html>
```

HTMLTableCellElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object is used to represent the TH and TD elements. The `HTMLTableCellElement` object inherits all methods and properties from the `HTMLElement` object. Table 11.35 contains a list of properties for this object.

Table 11.35 *Properties of the `HTMLTableCellElement` Object*

Item	Description
<code>abbr</code>	Abbreviation for header cells
<code>align</code>	Horizontal alignment for data in cell
<code>axis</code>	Names groups of related headers
<code>bgColor</code>	Background color for this cell
<code>cellIndex</code>	Index of this cell in the row
<code>ch</code>	Alignment character for cells in a column
<code>chOff</code>	Alignment character offset
<code>colSpan</code>	The number of columns that this cell spans
<code>headers</code>	Collection of headers for this cell
<code>height</code>	The height of this cell in pixels
<code>noWrap</code>	Indicates that text in this cell does not wrap if true
<code>rowSpan</code>	Number indicating how many rows this cell spans
<code>scope</code>	Scope covered by the header cells
<code>vAlign</code>	Vertical alignment of data in cells
<code>width</code>	Width of cell in pixels

Example

Listing 11.297 illustrates the creation of an `HTMLTableCellElement` using the `insertRow()` method of the `HTMLTableRowElement` object.

Listing 11.297 *Creating an* `HTMLTableCellElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
var cellObj = rowObj.insertCell(2);
// -->
</script>
</html>
```

HTMLTableCellElement.abbr

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.abbr`

Description

The `abbr` property of the `HTMLTableCellElement` object is an abbreviation for header cells.

Example

Listing 11.298 illustrates the creation of an `HTMLTableCellElement` using the `HTMLTableRowElement` object and then setting its `abbr` property.

Listing 11.298 *Creating an* `HTMLTableCellElement` *and Setting Its* **Property**

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
var cellObj = rowObj.insertCell(1);
cellObj.abbr = "Admin";
// -->
</html>
```

HTMLTableCellElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.align`

Description

The `align` property of the `HTMLTableCellElement` object is the horizontal alignment of data in cell.

Example

Listing 11.299 illustrates the creation of an `HTMLTableCellElement` using the `HTMLTableRow` object and then setting its `align` property.

Listing 11.299 *Creating an `HTMLTableCellElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.align = "center";
// -->
</script>
</html>
```

HTMLTableCellElement.axis

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.axis`

Description

The `axis` property of the `HTMLTableCellElement` object names group of related headers.

Example

Listing 11.300 illustrates the creation of an `HTMLTableCellElement` and then setting its `axis` property.

Listing 11.300 *Creating an `HTMLTableCellElement` and Setting Its `axis` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
```

```
<!--  
cellObj.axis = "Colors";  
// -->  
</script>  
</html>
```

HTMLTableCellElement.bgColor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.bgColor

Description

The bgColor property of the HTMLTableCellElement object is the cell background color.

Example

Listing 11.301 illustrates the creation of an HTMLTableCellElement and then setting its bgColor property.

Listing 11.301 *Creating an HTMLTableCellElement and Setting Its bgColor Property*

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
cellObj.bgColor = "blue";  
// -->  
</script>  
</html>
```

HTMLTableCellElement.cellIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.cellIndex

Description

The cellIndex property of the HTMLTableCellElement object is the index of this cell in the row, starting from 0. This index is in document tree order and not display order.

Example

Listing 11.302 illustrates the creation of an `HTMLTableCellElement` and then reading its `cellIndex` property.

Listing 11.302 *Creating an `HTMLTableCellElement` and Reading Its `cellIndex` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
if(cellObj.cellIndex == 1)
    handleFirstCell(cellObj);
// -->
</script>
</html>
```

HTMLTableCellElement.ch

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.ch`

Description

The `ch` property of the `HTMLTableCellElement` object is the alignment character for cells in a column.

Example

Listing 11.303 illustrates the creation of an `HTMLTableCellElement` and then setting its `ch` property.

Listing 11.303 *Creating an `HTMLTableCellElement` and Setting Its `ch` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.ch = ".";
// -->
</html>
```

HTMLTableCellElement.chOff

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.chOff`

Description

The `chOff` property of the `HTMLTableCellElement` object is the offset of alignment character.

Example

Listing 11.304 illustrates the creation of an `HTMLTableCellElement` and then setting its `chOff` property.

Listing 11.304 *Creating an `HTMLTableCellElement` and Setting Its `chOff` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.chOff = "1";
// -->
</script>
</html>
```

HTMLTableCellElement.colSpan

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.colSpan`

Description

The `colSpan` property of the `HTMLTableCellElement` object is the number of columns spanned by cell.

Example

Listing 11.305 illustrates the creation of an `HTMLTableCellElement` and then setting its `colSpan` property.

Listing 11.305 *Creating an `HTMLTableCellElement` and Setting `colSpan` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.colSpan = 3;
// -->
</script>
</html>
```

HTMLTableCellElement.headers

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.headers`

Description

The `headers` property of the `HTMLTableCellElement` object is a list of id attribute values for header cells.

Example

Listing 11.306 illustrates the creation of an `HTMLTableCellElement` and then setting its `headers` property.

Listing 11.306 *Creating an `HTMLTableCellElement` and Setting Its header Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.headers = "foo,bar";
// -->
</script>
</html>
```

HTMLTableCellElement.height

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.height`

Description

The `height` property of the `HTMLTableCellElement` object is the cell height.

Example

Listing 11.307 illustrates the creation of an `HTMLTableCellElement` and then setting its `height` property.

Listing 11.307 *Creating an `HTMLTableCellElement` and Setting Its height Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

```
cellObj.height = 25;  
// -->  
</script>  
</html>
```

HTMLTableCellElement.noWrap

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.noWrap

Description

The noWrap property of the HTMLTableCellElement object Suppresses word wrapping when true.

Example

Listing 11.308 illustrates the creation of an HTMLTableCellElement and then setting its noWrap property.

Listing 11.308 Creating an HTMLTableCellElement and Setting Its noWrap Property

```
<html>  
<script language="JavaScript" type="text/javascript">  
!!--  
cellObj.noWrap = true;  
// -->  
</script>  
</html>
```

HTMLTableCellElement.rowSpan

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.rowSpan

Description

The rowSpan property of the HTMLTableCellElement object is the number of rows spanned by cell.

Example

Listing 11.309 illustrates the creation of an HTMLTableCellElement and then setting its rowSpan property.

Listing 11.309 *Creating an* HTMLTableCellElement *and Setting Its* **rowSpan** *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.rowSpan = 3;
// -->
</script>
</html>
```

HTMLTableCellElement.scope

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.scope

Description

The scope property of the HTMLTableCellElement object is the scope covered by header cells.

Example

Listing 11.310 illustrates the creation of an HTMLTableCellElement and then setting its scope property.

Listing 11.310 *Creating an* HTMLTableCellElement *and Setting Its* **scope** *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.scope = "row";
// -->
</script>
</html>
```

HTMLTableCellElement.vAlign

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableCellObj.vAlign

Description

the vAlign property of the HTMLTableCellElement object is the vertical alignment of data in cell.

Example

Listing 11.311 illustrates the creation of an `HTMLTableCellElement` and then setting its `vAlign` property.

Listing 11.311 *Creating an `HTMLTableCellElement` and Setting Its `vAlign` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.vAlign = "top";
// -->
</script>
</html>
```

HTMLTableCellElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableCellObj.width`

Description

The `width` property of the `HTMLTableCellElement` object is the cell width.

Example

Listing 11.312 illustrates the creation of an `HTMLTableCellElement` and then setting its `width` property.

Listing 11.312 *Creating an `HTMLTableCellElement` and Setting Its `width` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
cellObj.width = 35;
// -->
</script>
</html>
```

HTMLTableColElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents both the HTML COL and COLGROUP elements. HTMLTableColElement object inherits all properties and methods from HTMLTableElement. Table 11.36 contains a list of properties for this object.

Table 11.36 *Properties of the HTMLTableColElement Object*

Item	Description
align	Alignment of cell data in column
ch	Alignment character for cells in a column
chOff	Offset of alignment character
span	Indicates the number of columns affected by grouping
vAlign	Vertical alignment of cell data in column
width	The width of the table column

Example

Listing 11.313 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object.

Listing 11.313 *Creating an HTMLTableColElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("td");

// -->
</script>
</html>
```

HTMLTableColElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTableColObj.align
```

Description

The The align property of the HTMLTableColElement object is the horizontal alignment of cell data in column.

Example

Listing 11.314 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object and then setting its align property.

Listing 11.314 *Creating an* HTMLTableColElement *and Setting Its* align *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("td");
columnObj.align = "center";
// -->
</script>
</html>
```

HTMLTableColElement.ch

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableColObject.ch

Description

The ch property of the HTMLTableColElement object is the alignment character for cells in a column.

Example

Listing 11.315 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object and then setting its ch property.

Listing 11.315 *Creating an* HTMLTableColElement *and Setting Its* ch *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("td");
columnObj.align = "char";
columnObj.ch = ".";
// -->
</script>
</html>
```

HTMLTableColElement.chOff

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableColObject.chOff

Description

The chOff property of the HTMLTableColElement object is the offset of alignment character.

Example

Listing 11.316 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object element and then setting its chOff property.

Listing 11.316 Setting the chOff Property of HTMLTableColElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("col");
columnObj.align = "char";
columnObj.chOff = 3;
// -->
</script>
</html>
```

HTMLTableColElement.span

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableColObj.span

Description

The span property of the HTMLTableColElement object indicates the number of columns in a group or affected by a grouping.

Example

Listing 11.317 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object and then setting its span property.

Listing 11.317 *Creating an* `HTMLTableColElement` *and Setting Its* `span` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("col");
columnObj.span = 3;
// -->
</script>
</html>
```

HTMLTableColElement.vAlign

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableColObj.vAlign`

Description

The `vAlign` property of the `HTMLTableColElement` object is the vertical alignment of cell data in column.

Example

Listing 11.318 illustrates the creation of an `HTMLTableColElement` using the `HTMLTableRowElement` object and then setting its `vAlign` property.

Listing 11.318 *Setting the* `vAlign` *Property of the* `HTMLTableColElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("td");
columnObj.vAlign = "top";
// -->
</script>
</html>
```

HTMLTableColElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableColObj.width`

Description

The width property of the HTMLTableColElement object is the default column width.

Example

Listing 11.319 illustrates the creation of an HTMLTableColElement using the HTMLTableRowElement object element and then setting its width property.

Listing 11.319 *Setting the width Property of the* HTMLTableColElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(2);
var columnObj = rowObj.createElement("td");
columnObj.width = 44;
// -->
</script>
</html>
```

HTMLTableElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML table element. HTMLTableElement inherits all properties and methods from the HTMLElement object. Table 11.37 contains a list of properties and methods for this object.

Table 11.37 *Contents of the* HTMLTableElement *Object*

Type	Item	Description
Property	align	The alignment of this table in relation to the rest of the document.
	bgColor	The background color for this table.
	border	The width of the border around the table.
	caption	The caption for this table element.
	cellPadding	Space between cell contents and cell borders.
	cellSpacing	Space between cells.
	frame	Specifies which external table borders to render.
	rows	Collection of all rows in the table.
	rules	Which internal table borders to render.
	summary	Description of the purpose of the table structure.
	tBodies	Collections of defined table bodies for this table.

Type	Item	Description
Method	tFoot	The table's TFOOT.
	tHead	The table's THEAD.
	width	The width of the table.
	createCaption()	Creates table caption or returns an existing one.
	createTFoot()	Creates a table footer or returns an existing one.
	createTHead()	Creates a table heading or returns an existing one.
	deleteCaption()	Deletes a table caption if one exists.
	deleteTFoot()	Deletes a table footer if one exists.
	deleteTHead()	Deletes a table heading if one exists.
	deleteRow()	Deletes a row from the table.
	insertRow()	Inserts a row into the table.

Example

Listing 11.320 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its caption property.

Listing 11.320 *Creating an `HTMLTableElement` and Setting Its caption Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.caption = "My Table";
// -->
</script>
</html>
```

HTMLTableElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.align`

Description

The `align` property of the `HTMLTableElement` object specifies the table's position with respect to the rest of the document.

Example

Listing 11.321 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its `align` property.

Listing 11.321 *Setting the align Property of the* HTMLTableElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.align = "top";
// -->
</script>
</html>
```

HTMLTableElement.bgColor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.bgColor

Description

The bgColor property of the HTMLTableElement object is the cell background color.

Example

Listing 11.322 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then setting its bgColor property.

Listing 11.322 *Creating an* HTMLTableElement *and Setting Its* bgColor Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.bgColor = "#000000";
// -->
</script>
</html>
```

HTMLTableElement.border

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.border

Description

The border property of the HTMLTableElement object is the width of the border around the table.

Example

Listing 11.323 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its `border` property.

Listing 11.323 *Setting the `border` Property of `HTMLTableElement`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.border = 1;
// -->
</script>
</html>
```

HTMLTableElement.caption

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.caption`

Description

The `caption` property of the `HTMLTableElement` object returns the table's `CAPTION`, or `void` if none exists.

Example

Listing 11.324 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its `caption` property.

Listing 11.324 *Creating an `HTMLTableElement` and Setting Its `caption` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.caption = "My Table";
// -->
</script>
</html>
```

HTMLTableElement.cellPadding

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.cellPadding

Description

The cellPadding property of the HTMLTableElement object specifies the horizontal and vertical space between cell content and cell borders.

Example

Listing 11.325 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then setting its cellPadding property.

Listing 11.325 *Creating an HTMLTableElement and Setting Its cellPadding Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.cellPadding = 0;
// -->
</script>
</html>
```

HTMLTableElement.cellSpacing

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.cellSpacing

Description

The cellSpacing property of the HTMLTableElement object specifies the horizontal and vertical separation between cells.

Example

Listing 11.326 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then setting its cellSpacing property.

Listing 11.326 *Creating an HTMLTableElement and Setting Its cellSpacing Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.cellSpacing = 2;
// -->
</script>
</html>
```

HTMLTableElement.createCaption()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.createCaption()

Description

The createCaption() method of the HTMLTableElement object creates a new table caption object or return an existing one.

Example

Listing 11.327 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then the invocation of its createCaption() method.

Listing 11.327 *Creating an HTMLTableElement and Setting Its createCaption() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var captionObj = tableObj.createCaption();
// -->
</script>
</html>
```

HTMLTableElement.createTFoot()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.createTFoot()

Description

The `createTFoot()` method of the `HTMLTableElement` object creates a table footer row or return an existing one.

Example

Listing 11.328 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object element and then the invocation of its `createTFoot()` method.

Listing 11.328 *Creating an `HTMLTableElement` and Invoking `createTFoot()`*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var tFootObj = tableObj.createTFoot();
// -->
</script>
</html>
```

HTMLTableElement.createTHead()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.createTHead()`

Description

The `createTHead()` method of the `HTMLTableElement` object

Example

Listing 11.329 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then the invocation of its `createTHead()` method.

Listing 11.329 *Creating an `HTMLTableElement` and Invoking Its `createTHead()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var tHeadObj = tableObj.createTHead();
// -->
</script>
</html>
```

HTMLTableElement.deleteCaption()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.deleteCaption()

Description

The deleteCaption() method of the HTMLTableElement object deletes the table caption, if one exists.

Example

Listing 11.330 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then the invocation of its deleteCaption() method.

Listing 11.330 *Creating an HTMLTableElement and Invoking Its deleteCaption() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.deleteCaption();
// -->
</script>
</html>
```

HTMLTableElement.deleteRow()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.deleteRow(*index*)

Description

The deleteRow() method of the HTMLTableElement object deletes the *index* table row.

Example

Listing 11.331 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then the invocation of its deleteRow() method.

Listing 11.331 *Invoking the deleteRow() Method of HTMLTableElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
```

Listing 11.331 Continued

```
var tableObj = bodyObj.createElement("table");
tableObj.deleteRow(1);
// -->
</script>
</html>
```

HTMLTableElement.deleteTFoot()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTableObj.deleteTFoot()
```

Description

The `deleteTFoot()` method of the `HTMLTableElement` object deletes the footer from the table, if one exists.

Example

Listing 11.332 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then the invocation of its `deleteTFoot()` method.

Listing 11.332 Creating an HTMLTableElement and Invoking Its deleteTFoot() Method

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.CreateElement("table");
tableObj.deleteTFoot();
// -->
</script>
</html>
```

HTMLTableElement.deleteTHead()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTableObj.deleteTHead()
```

Description

The `deleteTHead()` method of the `HTMLTableElement` object deletes the header from the table, if one exists.

Example

Listing 11.333 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then the invocation of its `deleteHead()` method.

Listing 11.333 *Creating an `HTMLTableElement` and Invoking Its `deleteHead()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.deleteHead();
// -->
</script>
</html>
```

HTMLTableElement.frame

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.frame`

Description

The `frame` property of the `HTMLTableElement` object specifies which external table borders to render.

Example

Listing 11.334 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its `frame` property.

Listing 11.334 *Creating an `HTMLTableElement` and Setting Its `frame` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.frame = "box";
// -->
</script>
</html>
```

HTMLTableElement.insertRow()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.insertRow(index)

Description

The `insertRow()` method of the `HTMLTableElement` object inserts a new empty row in the table. The new row is inserted immediately before and in the same section as the current *index* row in the table. If *index* is equal to the number of rows, the new row is appended. In addition, when the table is empty, the row is inserted into a `TBODY` that is created and inserted into the table.

Example

Listing 11.335 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then the invocation of its `insertRow()` method.

Listing 11.335 *Creating an HTMLTableElement and Invoking Its insertRow() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
var rowObj = tableObj.insertRow(1);
// -->
</script>
</html>
```

HTMLTableElement.rows

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObject.rows

Description

The `rows` property of the `HTMLTableElement` object returns a collection of all the rows in the table, including all in `THEAD`, `TFOOT`, and `TBODY` elements.

Example

Listing 11.336 illustrates the creation of an `HTMLTableElement` and then reading its `rows` property.

Listing 11.336 *Reading the rows Property of the HTMLTableElement Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowsCollection = tableObj.rows;
// -->
</script>
</html>
```

HTMLTableElement.rules

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.rules

Description

The rules property of the HTMLTableElement object specifies which internal table borders to render.

Example

Listing 11.337 illustrates the creation of an HTMLTableElement using the HTMLBodyElement object and then setting its rules property.

Listing 11.337 *Creating an HTMLTableElement and Setting Its rules Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.rules = "all";
// -->
</script>
</html>
```

HTMLTableElement.summary

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableObj.summary

Description

The summary property of the HTMLTableElement object is a description about the purpose or structure of a table.

Example

Listing 11.338 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its `summary` property.

Listing 11.338 *Creating an `HTMLTableElement` and Setting Its `summary` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.summary = "This table has no purpose";
// -->
</script>
</html>
```

HTMLTableElement.tBodies

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.tBodies`

Description

The `tBodies` property of the `HTMLTableElement` object returns a collection of the defined table bodies.

Example

Listing 11.339 illustrates the creation of an `HTMLTableElement` and then reading its `tBodies` property.

Listing 11.339 *Creating an `HTMLTableElement` and Reading Its `tBodies` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tBodiesCollection = tableObj.tBodies;
// -->
</script>
</html>
```

HTMLTableElement.tFoot

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.tFoot`

Description

The `tFoot` property of the `HTMLTableElement` object returns the table's `TFOOT`, or `Null` if none exists.

Example

Listing 11.340 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then reading its `tFoot` property.

Listing 11.340 *Creating an HTMLTableElement and Reading Its tFoot Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.tFoot;
// -->
</script>
</html>
```

HTMLTableElement.tHead

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.tHead`

Description

The `tHead` property of the `HTMLTableElement` object returns the table's `THEAD`, or `Null` if none exists.

Example

Listing 11.341 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then reading its `tHead` property.

Listing 11.341 *Creating an `HTMLTableElement` and Reading Its `tHead` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.tHead;
// -->
</script>
</html>
```

HTMLTableElement.width

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableObj.width`

Description

The width property of the `HTMLTableElement` object specifies the desired table width.

Example

Listing 11.342 illustrates the creation of an `HTMLTableElement` using the `HTMLBodyElement` object and then setting its width property.

Listing 11.342 *Creating an `HTMLTableElement` and Setting Its width Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableObj = bodyObj.createElement("table");
tableObj.width = "100%";
// -->
</script>
</html>
```

HTMLTableRowElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents a row in an HTML table element. `HTMLTableRowElement` inherits all methods and properties from `HTMLElement`. Table 11.38 contains a list of properties and methods for this object.

Table 11.38 Contents of the `HTMLTableRowElement` Object

Type	Item	Description
Property	<code>align</code>	Alignment of data within cells on this row.
	<code>bgColor</code>	Background color of cells on this row.
	<code>cells</code>	Collection of cells in this row.
	<code>ch</code>	Alignment character for cells on this row.
	<code>chOff</code>	Alignment character offset.
	<code>rowIndex</code>	Index of this row relative to the entire table.
	<code>sectionRowIndex</code>	Index of this row relative to the table section.
	<code>vAlign</code>	Vertical <code>vAlign</code> ment of cell data for this row.
Method	<code>deleteCell()</code>	Removes a cell from this row.
	<code>insertCell()</code>	Inserts a cell into this row.

Example

Listing 11.343 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object's `insertRow()` method.

Listing 11.343 Creating an `HTMLTableRowElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
// -->
</script>
</html>
```

HTMLTableRowElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableRowObj.align`

Description

The `align` property of the `HTMLTableRowElement` object is the horizontal alignment of data within cells of this row.

Example

Listing 11.344 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object and then setting its `align` property.

Listing 11.344 *Creating an `HTMLTableRowElement` and Setting Its `align` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
rowObj.align = "left";
// -->
</script>
</html>
```

HTMLTableRowElement.bgColor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableRowObj.bgColor`

Description

The `bgColor` property of the `HTMLTableRowElement` object is the background color for rows.

Example

Listing 11.345 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object and then setting its `bgColor` property.

Listing 11.345 *Creating an `HTMLTableRowElement` and Setting Its `bgColor` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
rowObj.bgColor = "red";
// -->
</script>
</html>
```

HTMLTableRowElement.cells

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.cells

Description

The cells property of the HTMLTableRowElement object is the collection of cells in this row.

Example

Listing 11.346 illustrates the creation of an HTMLTableRowElement and then reading its cells property.

Listing 11.346 *Creating an HTMLTableRowElement and Reading Its cells Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var tableCellsCollection = rowObj.cells;
// -->
</script>
</html>
```

HTMLTableRowElement.ch

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.ch

Description

The ch property of the HTMLTableRowElement object is the alignment character for cells in a column.

Example

Listing 11.347 illustrates the creation of an HTMLTableRowElement using the HTMLTableElement object and then setting its ch property.

Listing 11.347 *Creating an HTMLTableRowElement and Setting Its ch Property*

```
<html>
<script language="JavaScript" type="text/javascript">
```

Listing 11.347 Continued

```
<!--  
var rowObj = tableObj.insertRow(1);  
rowObj.align = "char";  
rowObj.ch = ".";  
// -->  
</script>  
</html>
```

HTMLTableRowElement.chOff

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.chOff

Description

The chOff property of the HTMLTableRowElement object is the offset of alignment character.

Example

Listing 11.348 illustrates the creation of an HTMLTableRowElement using the HTMLTableElement object and then setting its chOff property.

Listing 11.348 Creating an HTMLTableRowElement and Setting Its chOff Property

```
<html>  
<script language="JavaScript" type="text/javascript">  
<!--  
var rowObj = tableObj.insertRow(1);  
rowObj.chOff = "3";  
// -->  
</script>  
</html>
```

HTMLTableRowElement.deleteCell()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.deleteCell(*index*)

Description

The deleteCell() method of the HTMLTableRowElement object deletes a cell from the current row.

Example

Listing 11.349 illustrates the creation of an `HTMLTableRowElement` and then invokes its `deleteCell()` method.

Listing 11.349 *Creating an `HTMLTableRowElement` and Invoking Its `deleteCell()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
rowObj.deleteCell(1);
// -->
</script>
</html>
```

HTMLTableRowElement.insertCell()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTableRowObj.insertCell(index)
```

Description

The `insertCell()` method of the `HTMLTableRowElement` object inserted an empty TD cell into this row. If `index` is equal to the number of cells, the new cell is appended.

Example

Listing 11.350 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object and then the invocation of its `insertCell()` method.

Listing 11.350 *Invoking the `insertCell()` Method of the `HTMLTableRowElement` Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
var cellObj = rowObj.insertCell(2);
// -->
</script>
</html>
```

HTMLTableRowElement.rowIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.rowIndex

Description

The `rowIndex` property of the `HTMLTableRowElement` object is the index of this row, relative to the entire table, starting from 0. This is in document tree order and not display order. The `rowIndex` does not take into account sections (THEAD, TFOOT, or TBODY) within the table.

Example

Listing 11.351 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object element and then reading its `rowIndex` property.

Listing 11.351 *Creating an HTMLTableRowElement and Reading Its rowIndex Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
if(rowObj.rowIndex == 1)
    handleFirstRow(rowObj);
// -->
</script>
</html>
```

HTMLTableRowElement.sectionRowIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableRowObj.sectionRowIndex

Description

The `sectionRowIndex` property of the `HTMLTableRowElement` object is the index of this row, relative to the current section (THEAD, TFOOT, or TBODY), starting from 0.

Example

Listing 11.352 illustrates the creation of an `HTMLTableRowElement` and then reading its `sectionRowIndex` property.

Listing 11.352 *Creating an* `HTMLTableRowElement` *and Reading Its* `sectionRowIndex` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
if(rowObj.sectionRowIndex == 1)
    handleFirstSectionRow(rowObj);
// -->
</script>
</html>
```

HTMLTableRowElement.vAlign

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableRowObj.vAlign`

Description

The `vAlign` property of the `HTMLTableRowElement` object is the vertical alignment of data within cells of this row.

Example

Listing 11.353 illustrates the creation of an `HTMLTableRowElement` using the `HTMLTableElement` object and then setting its `vAlign` property.

Listing 11.353 *Creating an* `HTMLTableRowElement` *and Setting Its* `vAlign` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableObj.insertRow(1);
rowObj.vAlign = "top";
// -->
</script>
</html>
```

HTMLTableSectionElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents the HTML elements THEAD, TFOOT, and TBODY. The HTMLTableSectionElement object inherits all methods and properties from HTMLTableElement. Table 11.39 contains a list of properties and methods for this object.

Table 11.39 Contents of the HTMLTableSectionElement Object

Type	Item	Description
Property	align	Alignment of data in cells.
	ch	Alignment character for cells in a column.
	chOff	Alignment character offset.
	rows	Collection of rows in this table section.
	vAlign	Vertical alignment for data in cells.
Method	deleteRow()	Deletes a row from this table section.
	insertRow()	Inserts a row into this table section.

Example

Listing 11.354 illustrates the creation of an HTMLTableSectionElement and then setting its align property.

Listing 11.354 Creating an HTMLTableSectionElement and Setting Its align Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.align = "char";
// -->
</script>
</html>
```

HTMLTableSectionElement.align

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTableSectionObj.align
```

Description

The align property of the HTMLTableSectionElement object is the horizontal alignment of data in cells.

Example

Listing 11.355 illustrates the creation of an HTMLTableSectionElement object and then setting its align property.

Listing 11.355 *Creating an HTMLTableSectionElement and Setting Its align Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.align = "char";
// -->
</script>
</html>
```

HTMLTableSectionElement.ch

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableSection.ch

Description

The ch property of the HTMLTableSectionElement object is the alignment character for cells in a column.

Example

Listing 11.356 illustrates the creation of an HTMLTableSectionElement and then setting its ch property.

Listing 11.356 *Creating an HTMLTableSectionElement and Setting Its ch Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.align = "char";
tableSectionObj.ch = ".";
// -->
</script>
</html>
```

HTMLTableSectionElement.chOff

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTableSectionObj.chOff

Description

The chOff property of the HTMLTableSectionElement object is the offset of alignment character.

Example

Listing 11.357 illustrates the creation of an `HTMLTableSectionElement` object and then setting its `chOff` property.

Listing 11.357 *Creating an `HTMLTableSectionElement` and Setting Its `chOff` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.align = "char";
tableSectionObj.chOff = "22";
// -->
</script>
</html>
```

HTMLTableSectionElement.deleteRow()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableSectionObj.deleteRow(index)`

Description

The `deleteRow()` method of the `HTMLTableSectionElement` deletes a row from this section.

Example

Listing 11.358 illustrates the creation of an `HTMLTableSectionElement` object and then invoking its `deleteRow()` method.

Listing 11.358 *Creating an `HTMLTableSectionElement` and Invoking Its `deleteRow()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.deleteRow(1);
// -->
</script>
</html>
```

HTMLTableSectionElement.insertRow()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableSectionObj.insertRow(index)`

Description

The `insertRow()` of the `HTMLTableSectionElement` object inserts a row into this section. The new row is inserted immediately before the current *index* row in this section. If *index* is equal to the number of rows in this section, the new row is appended.

Example

Listing 11.359 illustrates the creation of an `HTMLTableSectionElement` and then the invocation of its `insertRow()` method.

Listing 11.359 *Creating an HTMLTableSectionElement and Invoking Its insertRow() Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowObj = tableSectionObj.insertRow(1);
// -->
</script>
</html>
```

HTMLTableSectionElement.rows

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableSectionObj.rows`

Description

The `rows` property of the `HTMLTableSectionElement` object is the collection of rows in this table section.

Example

Listing 11.360 illustrates the creation of an `HTMLTableSectionElement` and then reading its `rows` property.

Listing 11.360 *Reading the rows Property of the* `HTMLTableSectionElement` *Object*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var rowsCollection = tableSectionObj.rows;
// -->
</script>
</html>
```

HTMLTableSectionElement.vAlign

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTableSectionObj.vAlign`

Description

The `vAlign` property of the `HTMLTableSectionElement` object is the vertical alignment of data in cells.

Example

Listing 11.361 illustrates the creation of an `HTMLTableSectionElement` and then setting its `vAlign` property.

Listing 11.361 *Creating an* `HTMLTableSectionElement` *and Setting Its* `vAlign` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
tableSectionObj.align = "char";
tableSectionObj.vAlign = "top";
// -->
</script>
</html>
```

HTMLTextAreaElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML multi-line text field. `HTMLTextAreaElement` inherits all properties and methods from `HTMLElement`. Table 11.40 contains a list of properties and methods for this object.

Table 11.40 *Contents of the* `HTMLTextAreaElement` *Object*

Type	Item	Description
Property	<code>accessKey</code>	Single character access key that give control to this element.
	<code>cols</code>	Width of text area element in characters.
	<code>defaultValue</code>	The default value for this text area element.
	<code>disabled</code>	Indicates whether or not this text area element is disabled.
	<code>form</code>	The form in which this text area element resides.
	<code>name</code>	Indicates the name of this text area element.
	<code>readOnly</code>	Indicates whether or not the contents of this text area element are readonly.
	<code>rows</code>	Number of rows in this text area element.
	<code>tabIndex</code>	Number representing the tab order of this text area element.
	<code>type</code>	The string "text"area".
	<code>value</code>	The contents of this text area element.
	<code>blur()</code>	Removes keyboard focus from this text area element.
Method	<code>focus()</code>	Gives keyboard focus to this text area element.
	<code>select()</code>	Selects the contents of this text area element.

Example

Listing 11.362 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object.

Listing 11.362 *Creating an* `HTMLTextAreaElement`

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("text"area");
// -->
</script>
</html>
```

HTMLTextAreaElement.accessKey

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTextAreaObj.accessKey`

Description

The `accessKey` property of the `HTMLTextAreaElement` object is a single character access key to give access to the form control.

Example

Listing 11.363 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object element and then setting its `accessKey` property.

Listing 11.363 *Creating an `HTMLTextAreaElement` and Setting Its `accessKey` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.accessKey = "q";
// -->
</script>
</html>
```

HTMLTextAreaElement.blur()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlTextAreaObj.blur()
```

Description

The `blur()` method of the `HTMLTextAreaElement` object removes keyboard focus from this element.

Example

Listing 11.364 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object element and then the invocation of its `blur()` method.

Listing 11.364 *Creating an `HTMLTextAreaElement` and Invoking Its `blur()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.blur();
// -->
</script>
</html>
```

HTMLTextAreaElement.cols

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextArea.cols

Description

The cols property of the HTMLTextAreaElement object is the width of control (in characters).

Example

Listing 11.365 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then setting its cols property.

Listing 11.365 *Creating an HTMLTextAreaElement and Setting Its cols Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.cols = 25;
// -->
</script>
</html>
```

HTMLTextAreaElement.defaultValue

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.defaultValue

Description

The defaultValue of the HTMLTextAreaElement object represents the contents of the element. The value of this attribute doesn't change if the contents of the corresponding form control, in an interactive user agent, changes. Changing this attribute, however, resets the contents of the form control.

Example

Listing 11.366 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then setting its defaultValue property.

Listing 11.366 *Creating an* `HTMLTextAreaElement` *and Setting Its* `defaultValue` *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.defaultValue = "some default text";
// -->
</script>
</html>
```

HTMLTextAreaElement.disabled

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTextAreaObj.disabled`

Description

The disabled property of the `HTMLTextAreaElement` object is used to disable the `textarea` instance.

Example

Listing 11.367 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object and then setting its disabled property.

Listing 11.367 *Creating an* `HTMLTextAreaElement` *and Setting Its* disabled *Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.disabled = false;
// -->
</script>
</html>
```

HTMLTextAreaElement.focus()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTextAreaObj.focus()`

Description

The `focus()` method of the `HTMLTextAreaElement` object gives keyboard focus to this element.

Example

Listing 11.368 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object and then the invocation of its `focus()` method.

Listing 11.368 *Creating an `HTMLTextAreaElement` and Invoking Its `focus()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.focus();
// -->
</script>
</html>
```

HTMLTextAreaElement.form

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTextAreaObj.form`

Description

The `form` property of the `HTMLTextAreaElement` object returns the FORM element containing this control. Returns `Null` if this control isn't within the context of a form.

Example

Listing 11.369 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object and then reading its `form` property.

Listing 11.369 *Creating an `HTMLTextAreaElement` and Reading Its `form` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
if(textAreaObj.form.name == "emailForm")
    processEmailForm(textAreaObj.form);
// -->
</script>
</html>
```

HTMLTextAreaElement.name

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.name

Description

The name property of the HTMLTextAreaElement object is the form control or object name when submitted with a form.

Example

Listing 11.370 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then setting its name property.

Listing 11.370 *Creating an HTMLTextAreaElement and Setting Its name Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.name = "blurb";
// -->
</script>
</html>
```

HTMLTextAreaElement.readOnly

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.readOnly

Description

The readOnly property of the HTMLTextAreaElement object indicates whether or not this control is read-only.

Example

Listing 11.371 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object element and then setting its readOnly property.

Listing 11.371 *Creating an HTMLTextAreaElement and Setting Its readOnly Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.value = "Some readonly text";
textAreaObj.readOnly = true;
// -->
</script>
</html>
```

HTMLTextAreaElement.rows

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.rows

Description

The rows property of the HTMLTextAreaElement object is the number of text rows in the text area.

Example

Listing 11.372 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then setting its rows property.

Listing 11.372 *Creating an HTMLTextAreaElement and Setting Its rows Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.cols = 25;
textAreaObj.rows = 10;
// -->
</script>
</html>
```

HTMLTextAreaElement.select()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.select()

Description

The `select()` method of the `HTMLTextAreaElement` object selects the contents of the `TEXTAREA`.

Example

Listing 11.373 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object and then the invocation of its `select()` method.

Listing 11.373 *Creating an `HTMLTextAreaElement` and Invoking Its `select()` Method*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.value = "some text";
textAreaObj.select();
// -->
</script>
</html>
```

HTMLTextAreaElement.tabIndex

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`htmlTextAreaObj.tabIndex`

Description

The `tabIndex` property of the `HTMLTextAreaElement` object is the index that represents the element's position in the tabbing order.

Example

Listing 11.374 illustrates the creation of an `HTMLTextAreaElement` using the `HTMLFormElement` object and then setting its `tabIndex` property.

Listing 11.374 *Creating an `HTMLTextAreaElement` and Setting Its `tabIndex` Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.tabIndex = 3;
// -->
</script>
</html>
```


HTMLTextAreaElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.type

Description

The type property of the HTMLTextAreaElement object is the type of this form control. This is the string "textarea".

Example

Listing 11.375 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then reading its type property.

Listing 11.375 *Creating an HTMLTextAreaElement and Reading Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
display(textAreaObj.type);
// -->
</script>
</html>
```

HTMLTextAreaElement.value

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTextAreaObj.value

Description

The value property of the HTMLTextAreaElement object represents the current contents of the corresponding form control, in an interactive user agent. Changing this attribute changes the contents of the form control, but doesn't change the contents of the element. If the entirety of the data cannot fit into a single string, the implementation might truncate the data.

Example

Listing 11.376 illustrates the creation of an HTMLTextAreaElement using the HTMLFormElement object and then setting its value property.

Listing 11.376 *Creating an* HTMLTextAreaElement *and Setting Its* value Property

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var textAreaObj = formObj.createElement("textarea");
textAreaObj.value = "Some random text";
// -->
</script>
</html>
```

HTMLTitleElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents the document title. HTMLTitleElement inherits all properties and methods from HTMLElement. The property for this object is as follows:

<i><u>Item</u></i>	<i><u>Description</u></i>
text	The specified title as a string

Example

Listing 11.377 illustrates the creation of an HTMLTitleElement using the HTMLHeadElement object element.

Listing 11.377 *Creating an* HTMLTitleElement

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var headElement = htmlDoc.createElement("head");
var titleElement = htmlDoc.createElement("title");

// -->
</script>
</html>
```

HTMLTitleElement.text

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlTitleObj.text

Description

The title property of the HTMLTitleElement object represents the specified title of an HTML document.

Example

Listing 11.378 illustrates the creation of an HTMLTitleElement using the HTMLHeadElement object element and then setting its text property.

Listing 11.378 *Creating an HTMLTitleElement and Setting Its text Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var htmlDoc = HTMLDOMImplementation.createHTMLDocument("My HTML Doc");
htmlDoc.open();
var headElement = htmlDoc.createElement("head");
var titleElement = htmlDoc.createElement("title");
titleElement.text = "My New HTML Doc Title";
// -->
</script>
</html>
```

HTMLULListElement

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Core DOM HTML object.

Description

This object represents an HTML unordered list element. HTMLULListElement inherits all properties and method of HTMLDivElement. Table 11.41 contains a list of properties for this object.

Table 11.41 *Properties of the HTMLULListElement Object*

Item	Description
compact	Reduces spacing between list items if true
type	The bullet style of this unordered list element

Example

Listing 11.379 illustrates the creation of an HTMLULListElement using the HTMLBodyElement.

Listing 11.379 *Creating an HTMLULListElement*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var ulObj = bodyObj.CreateElement("ul");
// -->
</script>
</html>
```

HTMLULListElement.compact

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
htmlULListObj.compact
```

Description

The compact property of the HTMLULListElement object reduces spacing between list items.

Example

Listing 11.380 illustrates the creation of an HTMLULListElement using the HTMLBodyElement object and then setting its compact property.

Listing 11.380 *Creating an HTMLULListElement and Setting Its compact Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var ulObj = bodyObj.CreateElement("ul");
ulObj.compact = true;
// -->
</script>
</html>
```

HTMLUListElement.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

htmlUListObj.type

Description

The type property of the HTMLUListElement object is the bullet style of the element.

Example

Listing 11.381 illustrates the creation of an HTMLUListElement using the HTMLBodyElement object and then setting its type property.

Listing 11.381 *Creating an HTMLUListElement and Setting Its type Property*

```
<html>
<script language="JavaScript" type="text/javascript">
<!--
var ulObj = bodyObj.CreateElement("ul");
ulObj.type = "bullet";
// -->
</script>
</html>
```


CHAPTER 12

DOM Events

The DOM Level 2 Event Model was designed with two main goals. The first goal was to offer a generic model that would allow the registration of event handlers, flow for events to follow, and basic contextual information regarding the event. The second goal was to provide backward compatibility with the event models of DOM Level 0 and Level 1.

DocumentEvent

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The `DocumentEvent` object provides a mechanism by which the user can create an Event of a type supported by the implementation. It is expected that the `DocumentEvent` interface will be implemented on the same object which implements the `Document` interface in an implementation that supports the Event model. Contents of the `EventException` object are as follows:

<i>Type</i>	<i>Item</i>	<i>Description</i>
Method	<code>createEvent()</code>	Creates a new event.

Example

Listing 12.1 shows the creation of a mouse event object using the `createEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method.

Listing 12.1 *Creating and Initializing a Mouse Event Using the `createEvent()` Method of the `DocumentEvent` Object*

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initEvent("MouseEvent", true, false);
// -->
</script>
</html>
```

DocumentEvent.createEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`DocumentEvent.createEvent(eventType)`

Description

The `createEvent()` method of the `DocumentEvent` object creates a new event of the specified *eventType*.

Example

Listing 12.2 shows the creation of a mouse event object using the `CreateEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method.

Listing 12.2 *Creating and Initializing a Mouse Event Using the `createEvent()` Method of the `DocumentEvent` Object*

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initEvent("MouseEvent", true, false);
// -->
</script>
</html>
```

Event

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The Event interface is used to provide contextual information about an event to the handler processing the event. An object that implements the Event interface is generally passed as the first parameter to an event handler. More specific context information is passed to event handlers by deriving additional interfaces from Event, which contain information directly relating to the type of event they accompany. The object passed to the event listener also implements these derived interfaces.

An Event is created by using the `createEvent()` method of the DocumentEvent object. Table 12.1 lists all constants, methods and properties of the Event object.

Table 12.1 *Contents of Event Object*

Type	Item	Description
Constant	CAPTURING_PHASE	The phase of event flow when the event originates.
	AT_TARGET	The phase of event flow when the event is being evaluated at the target.
	BUBBLING_PHASE	The phase of event flow when the event is bubbling.
Property	bubbles	A Boolean indicating if this event has bubble capabilities.
	cancelable	A Boolean indicating if this event can be cancelled.
	currentTarget	An EventTarget indicating the current target for this event.
	eventPhase	A number indicating which phase this event is currently in.
	target	An EventTarget indicating the target for this event.
	timeStamp	A Date object indicating the time at which this event initiated.
	type	String indicating the type of this event.
Method	initEvent()	Initializes the current event.
	preventDefault()	Causes the current event to be cancelled thereby preventing its default flow from occurring.
	stopPropagation()	Prevents the event from propagating any further.

Example

Listing 12.3 illustrates the creation of an event using the `createEvent()` method of the DocumentEvent object.

Listing 12.3 *Creating an Event Object*

```
<html>
<script language="JScript">
<!--

var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initMouseEvent();
// -->
</script>
</html>
```

Event.type

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

eventObj.type

Description

The type property of the Event object identifies the name of the event.

Example

Listing 12.4 illustrates the creation of an EventListener function and checks the type property of an Event to see how it should be handled.

Listing 12.4 *Reading the type Property of the Event Object*

```
<html>
<script language="JScript">
<!--

function GenericEventHandler(evt) {
    if(evt.type. == "MouseEvent") {
        processMouseEvent(evt);
    } else if(evt.type == "MutationEvent") {
        processMutationEvent(evt);
    }
}
// -->
</script>
</html>
```

Event.bubbles

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`eventObj.bubbles`

Description

The bubbles property of the Event object is of type boolean and indicates whether this event can bubble.

Example

Listing 12.5 illustrates the creation of an EventListener function and checks the bubbles property of the evt parameter to determine whether the event bubbles.

Listing 12.5 *Reading the bubbles Property of the Event Object*

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    if(evt.bubbles)
        handleBubblesEvent(evt);
}
// -->
</script>
</html>
```

Event.cancelable

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`eventObj.cancelable`

Description

Used to indicate whether an event can have its default action prevented. If the default action can be prevented, the value is true; otherwise the value is false.

Example

Listing 12.6 illustrates the creation of an EventListener function and checks the cancelable property of the evt parameter to determine whether the event's default action can be canceled.

Listing 12.6 *Reading the cancelable Property of the Event Object*

```
<html>
<script language="JScript">
<!--
```

Listing 12.6 Continued

```
function GenericEventHandler(evt) {  
    if(evt.cancelable)  
        handleCancelableEvent(evt);  
}  
// -->  
</script>  
</html>
```

Event.currentTarget

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

eventObj.currentTarget

Description

The target property of an Event object indicates the EventTarget object to which this event is currently being dispatched.

Example

Listing 12.7 illustrates the creation of an EventListener function and checks the currentTarget property of the Event object to see how the event should be handled.

Listing 12.7 Reading the currentTarget Property of the Event Object

```
<html>  
<script language="JScript">  
!- -  
function GenericEventHandler(evt) {  
    var currentTarget = evt.currentTarget;  
    if(currentTarget.type == "TextArea")  
        handleTextAreaEvent(target);  
}  
// -->  
</script>  
</html>
```

Event.eventPhase

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

eventObj.eventPhase

Description

Indicates which phase of the event flow is being evaluated: `Event.CAPTURING_PHASE`, `Event.AT_TARGET`, or `Event.BUBBLING_PHASE`.

Example

Listing 12.8 illustrates the creation of an `EventListener` function and checks the `eventPhase` property of the `evt` parameter to determine at what phase the event is in.

Listing 12.8 *Reading the `eventPhase` Property of the Event Object*

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    if(evt.eventPhase == Event.bubblingPhase)
        processEventAtBubblingPhase(evt);
}
// -->
</script>
</html>
```

Event.initEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`eventObj.initEvent(eventType, canBubble, cancelable)`

Description

The `initEvent()` method of the `Event` object is used to initialize the value of an `Event` created using the `createEvent()` of the `DocumentEvent` object. This method can only be called before the `Event` has been dispatched via the `dispatchEvent()` method, although it can be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence. If called from a subclass of the `Event` interface, only the values specified in the `initEvent()` method are modified: All other attributes are left unchanged. The *eventType* parameter is a string that specifies the event type, *canBubble* specifies whether this event can bubble, and *cancelable* specifies whether the event's default action can be prevented.

Example

Listing 12.9 shows the creation of a mouse event object using the `CreateEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method.

Listing 12.9 *Creating and Initializing a Mouse Event Using the `initEvent()` Method of the Event Object*

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initEvent("MouseEvent", true, false);
// -->
</script>
</html>
```

Event.preventDefault()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
eventObj.preventDefault()
```

Description

If an event is cancelable, the `preventDefault()` method of the Event object is used to signify that the event is to be canceled, meaning any default action normally taken by the implementation as a result of the event will not occur. If, during any stage of event flow, the `preventDefault()` method is called, the event is canceled. Any default action associated with the event will not occur. Calling this method for a non-cancelable event has no effect. After `preventDefault()` has been called, it will remain in effect throughout the remainder of the event's propagation. This method can be used during any stage of event flow.

Example

Listing 12.10 illustrates the prevention of the default behavior if an event is cancelable.

Listing 12.10 *Invoking the `preventDefault()` Method of the Event Object*

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    if(evt.cancelable)
        evt.preventDefault();
}
// -->
</script>
</html>
```

Event.stopPropagation()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
eventObj.stopPropagation()
```

Description

The `stopPropagation()` method of the `Event` object is used prevent further propagation of an event during event flow. If any `EventListener` calls this method, the event will cease propagating through the tree. The event will complete dispatch to all listeners on the current `EventTarget` before event flow stops. This method can be used during any stage of event flow.

Example

Listing 12.11 illustrates the stoppage of the event flow of an event currently in the `AT_TARGET` phase.

Listing 12.11 Calling the stopPropagation() Method of the Event Object

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    if(evt.eventPhase == Event.AT_TARGET)
        evt.stopPropagation();
}
// -->
</script>
</html>
```

Event.target

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
eventObj.target;
```

Description

The `target` property of the `Event` object indicates the `EventTarget` object to which this event was originally dispatched.

Example

Listing 12.12 illustrates the creation of an `EventListener` function and checks the `target` property of the `Event` object to see how the event should be handled.

Listing 12.12 *Reading the target Property of the Event Object*

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    var target = evt.target;
    if(target.type == "TextArea")
        handleTextAreaEvent(target);
}
// -->
</script>
</html>
```

Event.timeStamp

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

eventObj.timeStamp

Description

The `timeStamp` property of the Event object is used to specify the time (in milliseconds relative to the epoch) at which the event was created. Because some systems might not provide this information, the value of `timeStamp` might not be available for all events. When not available, a value of `0` will be returned. Examples of epoch time are the time of the system start or 0:0:0 UTC 1st January 1970. The `timeStamp` property is an object of type Date.

Example

Listing 12.13 illustrates the creation of an event handler that behaves differently depending on the time when the event originated.

Listing 12.13 *Reading the timeStamp Property of the Event Object*

```
<html>
<script language="JScript">
<!--
function GenericEventHandler(evt) {
    if(evt.timeStamp < '11/24/2000')
        handleTS(evt.timeStamp);
}
// -->
</script>
</html>
```

EventException

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The EventException object is thrown when an exception condition is met while evaluating certain methods of other DOM Event objects. Table 12.2 lists the constants and properties for the EventException object.

Table 12.2 *Contents of EventException Object*

Type	Item	Description
Constant	UNSPECIFIED_EVENT_TYPE_ERR	The exception that occurs when the Event type is not defined.
Property	code	Indicates the type of error generated.

Example

Listing 12.14 illustrates the catching of an EventException object and reading its code property.

Listing 12.14 *Declaring an EventListener Function*

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}
try {
    var htmlDocument =
        ↳HTMLDOMImplementation.createHTMLDocument("My HTML Page");
    var textArea = htmlDocument.createElement("TextArea");
    textArea.addEventListener("MouseEvent", MouseEventHandler ,false);

    var mouseEvent = DocumentEvent.createEvent("MouseEvent");
    mouseEvent.initMouseEvent();
    textArea.dispatchEvent(mouseEvent);

}catch(evt) {
    if(evt.code == EventException.UNSPECIFIED_EVENT_TYPE_ERR)
        handleUnexpectedEvent(evt);
}
```

Listing 12.14 Continued

```
// -->
</script>
</html>
```

EventException.code

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

eventExceptionObj.code

Description

The code property of the EventException object contains the numeric code representing the actual exception.

Example

Listing 12.15 illustrates the catching of an EventException object and reading its code property.

Listing 12.15 Declaring an EventListener Function

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}

try {
    var htmlDocument =
        ↪HTMLDOMImplementation.createHTMLDocument("My HTML Page");
    var textArea = htmlDocument.createElement("TextArea");
    textArea.addEventListener("MouseEvent", MouseEventHandler ,false);

    var mouseEvent = DocumentEvent.createEvent("MouseEvent");
    mouseEvent.initMouseEvent();
    textArea.dispatchEvent(mouseEvent);

}catch(evt) {
    if(evt.code == EventException.UNSPECIFIED_EVENT_TYPE_ERR)
        handleUnexpectedEvent(evt);
}
```

```
// -->
</script>
</html>
```

EventListener

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The EventListener is simply a function reference that takes an object of type Event as an argument. When the event listener is registered, the body of the function will be executed when an specified type of event occurs.

Example

Listing 12.16 shows the declaration of an EventListener and then illustrates the registration of the listener with a TextArea object.

Listing 12.16 *Declaring an EventListener Function*

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}

var htmlDocument =
    ↪HTMLDOMImplementation.createHTMLDocument("My HTML Page");
var textArea = htmlDocument.createElement("TextArea");
textArea.addEventListener("MouseEvent", MouseEventHandler ,false);

var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initMouseEvent();

textArea.dispatchEvent(mouseEvent);
// -->
</script>
</html>
```

EventTarget

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

Specified by an Event object's *target* attribute, the EventTarget represents the object that an event is directed to by the DOM implementation. When the event reaches the target, all appropriate EventListeners are triggered.

All Node objects from a DOM implementation that supports the DOM Level 2 Event Model inherit from EventTarget. Table 12.1 shows all methods available for the EventTarget object. Table 12.3 lists all of the methods of the EventTarget object.

Table 12.3 Contents of the EventTarget Object

Type	Item	Description
Method	addEventListener()	Adds an event listener to this target.
	dispatchEvent()	Directs an event to this event target.
	removeEventListener()	Removes a specified event listener from this target.

Example

Listing 12.17 demonstrates the assignment of an event target and fires off an event.

Listing 12.17 Assigning and Sending an Event Using the dispatchEvent() Method of the EventTarget Object

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}

var htmlDocument = HTMLDOMImplementation.createHTMLDocument("My HTML Page");
var textArea = htmlDocument.createElement("TextArea");
textArea.addEventListener("MouseEvent", MouseEventHandler ,false);
```

```
var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initMouseEvent();

textArea.dispatchEvent(mouseEvent);
// -->
</script>
```

EventTarget.addEventListener()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
targetObj.addEventListener(type, listener, useCapture);
```

Description

The `addEventListener()` method of the `EventTarget` object adds an event listener to this target. *Type* is the event type to be listened for, the *listener* parameter is the actual listener function call, and *useCapture* is a Boolean field that determines whether the event can be intercepted by an event listener found in an ancestor of the target.

Example

Listing 12.18 demonstrates the assignment of an event listener and fires off an event.

Listing 12.18 Assigning and Sending an Event Using the addEventListener() Method of the EventTarget Object

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}

var htmlDocument = HTMLDOMImplementation.createHTMLDocument("My HTML Page");
var textArea = htmlDocument.createElement("TextArea");
textArea.addEventListener("MouseEvent", MouseEventHandler ,false);

var mouseEvent = DocumentEvent.createEvent("MouseEvent");
mouseEvent.initMouseEvent();
```

Listing 12.18 Continued

```
textArea.dispatchEvent(mouseEvent);  
// -->  
</script>  
</html>
```

EventTarget.dispatchEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
eventTargetObj.dispatchEvent(evt);
```

Description

The `dispatchEvent()` method of the `EventTarget` object directs the event `evt` to this `EventTarget` object.

Example

Listing 12.19 demonstrates using the `dispatchEvent()` method to direct a mouse event to an HTML `TextArea` object.

Listing 12.19 Dispatching a MouseEvent Using the dispatchEvent() Method of the EventTarget Object

```
<html>  
<script language="JScript">  
<!--  
function MouseEventHandler(evt) {  
    if(evt.type == "MouseEvent") {  
        if(evt.clientX ==5) && (evt.clientY ==5)  
            showDialog();  
    }  
}  
  
var htmlDocument = HTMLDOMImplementation.createHTMLDocument("My HTML Page");  
var textArea = htmlDocument.createElement("TextArea");  
textArea.addEventListener("MouseEvent", MouseEventHandler ,false);  
  
var mouseEvent = DocumentEvent.createEvent("MouseEvent");  
mouseEvent.initMouseEvent();  
  
textArea.dispatchEvent(mouseEvent);  
// -->  
</script>  
</html>
```


EventTarget.removeEventListener()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

```
eventListenerObj.removeEventListener(type, listener, useCapture)
```

Description

The `removeEventListener()` method of the `EventTarget` object removes the described and referenced listener from the list of event listeners maintained by this event target. *type* is the event type to being listened for, the *listener* parameter is the actual listener function call, and *useCapture* is a Boolean field that determines whether the event can be intercepted by an event listener found in an ancestor of the target.

Example

Listing 12.20 demonstrates removing an event listener from the first `TextArea` object that appears in an HTML document.

Listing 12.20 Removing an Event Listener Using the
`removeEventListener()` Method of the `EventTarget` Object

```
<html>
<script language="JScript">
<!--
function MouseEventHandler(evt) {
    if(evt.type == "MouseEvent") {
        if(evt.clientX ==5) && (evt.clientY ==5)
            showDialog();
    }
}

var htmlDocument = HTMLDOMImplementation.createHTMLDocument("My HTML Page");
var textAreaList = htmlDocument.getElementsByTagName("TextArea");
var firstTextArea = textAreaList.index(0);
firstTextArea.removeEventListener("MouseEvent", MouseEventHandler, false);
// -->
</script>
</html>
```

MouseEvent

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The `MouseEvent` object provides specific contextual information associated with mouse events. The `MouseEvent` object inherits all properties and methods from the `UIEvent` object. Table 12.4 lists all of the properties and methods of the `MouseEvent` object.

Table 12.4 Contents of `MouseEvent` Object

Type	Item	Description
Property	<code>altKey</code>	A Boolean that indicates whether the Alt key was pressed when this event occurred.
	<code>button</code>	A number that indicates whether a mouse button was pressed when this event occurred.
	<code>clientX</code>	The horizontal coordinate of the mouse event within the client area.
	<code>clientY</code>	The vertical coordinate of the mouse event within the client area.
	<code>ctrlKey</code>	A Boolean that indicates whether the CTRL key was pressed when this event occurred.
	<code>metaKey</code>	A Boolean that indicates whether a meta key was pressed when this event occurred.
	<code>relatedTarget</code>	Identifies a secondary target for this mouse event.
	<code>screenX</code>	The horizontal coordinate of mouse event.
	<code>screenY</code>	The vertical coordinate of the mouse event.
	<code>shiftKey</code>	A Boolean that indicates whether the Shift key was pressed when this event occurred.
Method	<code>initMouseEvent()</code>	Initializes a newly created <code>MouseEvent</code> object.

Example

Listing 12.21 shows the creation of a mouse event object using the `createEvent()` method of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method.

Listing 12.21 Creating and Initializing a `MouseEvent`

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.createEvent("MouseEvent");
```

```
mouseEvent.initMouseEvent("click", true, false, );  
// -->  
</script>  
</html>
```

MouseEvent.altKey

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.altKey

Description

Used to indicate whether the Alt key was depressed during the firing of the event. On some platforms, this key might map to an alternative key name.

Example

Listing 12.22 illustrates an event listener method as well as inspecting the altKey property if the event is a "click" mouse event.

Listing 12.22 Event Listener Inspecting altKey

```
<html>  
<script language="JScript">  
!- -  
function handleMouseEvent(evt) {  
    if((evt.type == "click") && (evt.altKey))  
        processClickAndCTRL(evt);  
// -->  
</script>  
</html>
```

MouseEvent.button

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.Button

Description

During mouse events caused by the depression or release of a mouse button, button is used to indicate which mouse button changed state. The values for button range from 0 to indicate the left button of the mouse, 1 to indicate the middle button if present, and 2 to indicate the right button. In mice configured for left-handed use in which the button actions are reversed, the values are instead read from right to left.

Example

Listing 12.23 illustrates an event listener method as well as inspecting which mouse button was pressed if the event is a "click" mouse event.

Listing 12.23 Event Listener Inspecting the button Property of the MouseEvent Object

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if((evt.type == "click") && (evt.button == 0))
        processLeftMouseClicked(evt);
    if((evt.type == "click") && (evt.button == 1))
        processMiddleMouseClicked(evt);
    if((evt.type == "click") && (evt.button == 2))
        processRightMouseClicked(evt);
// -->
</script>
</html>
```

MouseEvent.clientX

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.clientX

Description

The `clientX` property of the `MouseEvent` object represents the horizontal coordinate at which the event occurred relative to the DOM implementation's client area.

Example

Listing 12.24 illustrates an event listener method as well as inspecting the `clientX` property if the event is a "click" mouse event.

Listing 12.24 Event Listener Inspecting clientX

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if(evt.type == "click")
        processXLocation(evt.clientX);
}
// -->
</script>
</html>
```

MouseEvent.clientY

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mouseEventObj.clientY`

Description

The `clientY` property of the `MouseEvent` object represents the vertical coordinate at which the event occurred relative to the DOM implementation's client area.

Example

Listing 12.25 illustrates an event listener method as well as inspecting the `clientY` property if the event is a "click" mouse event.

Listing 12.25 *Event Listener Inspecting* `clientY`

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if(evt.type == "click")
        processXLocation(evt.clientY);
}
// -->
</script>
</html>
```

MouseEvent.ctrlKey

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mouseEventObj.ctrlKey`

Description

Used to indicate whether the CTRL key was depressed during the firing of the event.

Example

Listing 12.26 illustrates an event listener method as well as inspecting the `ctrlKey` property if the event is a "click" mouse event.

Listing 12.26 Event Listener Inspecting ctrlKey

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if((evt.type == "click") && (evt.ctrlKey))
        processClickAndCTRL(evt);
// -->
</script>
</html>
```

MouseEvent.initMouseEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.initMouseEvent(*type*, canBubble, cancelable, view, detail, screenX, screenY, clientX, clientY, ctrlKey, altKey, shiftKey, metaKey, button, relatedTarget)

Description

The `initMouseEvent()` method of the `MouseEvent` object is used to initialize the value of a `MouseEvent` created through the `DocumentEvent` object. This method can only be called before the `MouseEvent` has been dispatched via the `dispatchEvent()` method, although it can be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

The following types of mouse events are supported by DOM Level 2: click, mouseup, mousedown, mouseover, mousemove, and mouseout.

Example

Listing 12.27 shows the creation of a mouse event object using the `createEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method to initialize it.

Listing 12.27 Creating and Initializing a Mouse Event

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.CreateEvent("MouseEvent");
mouseEvent.initMouseEvent("click", true, false, );
// -->
</script>
</html>
```

MouseEvent.metaKey

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.metaKey

Description

The metaKey property of the MouseEvent object is used to indicate whether the META key was depressed during the firing of the event. On some platforms, this key might map to an alternative key name.

Example

Listing 12.28 illustrates an event listener method as well as inspecting the metaKey property if the event is a "click" mouse event.

Listing 12.28 *Event Listener Inspecting metaKey*

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if((evt.type == "click") && (evt.metaKey))
        processClickAndCTRL(evt);
// -->
</script>
</html>
```

MouseEvent.relatedTarget

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEvent.relatedTarget

Description

The relatedTarget property of the MouseEvent object is used to identify a secondary EventTarget related to a UI event. Currently, this attribute is used with the mouseover event to indicate the EventTarget, which the pointing device exited, and with the mouseout event to indicate the EventTarget, which the pointing device entered.

Example

Listing 12.29 illustrates reading the relatedTarget property of the MouseEvent object.

Listing 12.29 Event Listener Inspecting relatedTarget

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if(evt.relatedTarget == otherTarget
        processSecondaryTarget(otherTarget);
}
// -->
</script>
</html>
```

MouseEvent.screenX

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.screenX

Description

The screenX property of the MouseEvent object represents the horizontal coordinate at which the event occurred relative to the origin of the screen coordinate system.

Example

Listing 12.30 illustrates an event listener method as well as inspecting the screenX property if the event is a "click" mouse event.

Listing 12.30 Event Listener Inspecting screenX

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if(evt.type == "click")
        processXLocation(evt.ScreenX);
}
// -->
</script>
</html>
```

MouseEvent.screenY

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

mouseEventObj.screenY

Description

The `screenY` property of the `MouseEvent` object represents the vertical coordinate at which the event occurred relative to the origin of the screen coordinate system.

Example

Listing 12.31 illustrates an event listener method as well as inspecting the `screenY` property if the event is a "click" mouse event.

Listing 12.31 *Event Listener Inspecting* `screenY`

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if(evt.type == "click")
        processXLocation(evt.screenY);
}
// -->
</script>
</html>
```

MouseEvent.shiftKey

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mouseEventObj.shiftKey`

Description

The `shiftKey` property of the `MouseEvent` object is used to indicate whether the Shift key was depressed during the firing of the event.

Example

Listing 12.32 illustrates an event listener method as well as inspecting the `shiftKey` property if the event is a "click" mouse event.

Listing 12.32 *Event Listener Inspecting* `shiftKey`

```
<html>
<script language="JScript">
<!--
function handleMouseEvent(evt) {
    if((evt.type == "click") && (evt.shiftKey))
        processClickAndSHIFT(evt);
}
// -->
</script>
</html>
```

MutationEvent

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The `MutationEvent` object is designed to allow notification of any changes to the structure of a document, including `Attr` and `Text` modifications. It might be noted that none of the mutation events listed are designated as cancelable. This stems from the fact that it is very difficult to make use of existing DOM objects that cause document modifications if any change to the document might or might not take place because of the cancellation of the related event. Although this is still a desired capability, it was decided that it would be better left until the addition of transactions into the DOM. Many single modifications of the tree can cause multiple mutation events to be fired. Rather than attempt to specify the ordering of mutation events because of every possible modification of the tree, the ordering of these events is left to the implementation.

A DOM application can use the `hasFeature()` method of the `DOMImplementation` object with parameter values `MutationEvents` and `2.0` (respectively) to determine whether the mutation event module is supported by the implementation. Table 12.5 lists all of the constants, properties, and methods of the `MutationEvent` object.

Table 12.5 *Contents of the `MutationEvent` Object*

Type	Item	Description
Constant	MODIFICATION	Indicates whether this event occurred as a result of a modification.
	ADDITION	Indicates whether or not this event occurred as a result of an addition.
	REMOVAL	Indicates whether or not this event occurred as a result of a removal.
Property	attrChange	Indicates the attribute that was changed by this event.
	attrName	The attribute name that was associated with this event.
	newValue	The new value of this event.
	prevValue	The previous value of this event.
	relatedNode	The node related to this event.
Method	initMutationEvent()	Initializes a newly created <code>MutationEvent</code> object.

Example

Listing 12.33 shows the creation of a mutation event object using the `createEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method to initialize it.

Listing 12.33 Creating and Initializing a Mutation Event

```
<html>
<script language="JScript">
<!--
var mutationEvent = DocumentEvent.createEvent("MutationEvent");
mutationEvent.initEvent("DOMNodeInserted", true, false, relatedNode,
prevValue, newValue, attrName, attrChange);
// -->
</script>
</html>
```

MutationEvent.relatedNode

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEventObj.relatedNode`

Description

The `relatedNode` property of the `MutationEvent` object is used to identify a secondary node related to a mutation event. For example, if a mutation event is dispatched to a node indicating that its parent has changed, the `relatedNode` is the changed parent. If an event is instead dispatched to a subtree indicating a node was changed within it, the `relatedNode` is the changed node. In the case of the `DOMAttrModified` event it indicates the `Attr` node which was modified, added, or removed.

Example

Listing 12.34 shows the creation of a mutation event object using the `createEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method to initialize it.

Listing 12.34 Creating and Initializing a Mutation Event

```
<html>
<script language="JScript">
<!--
var mutationEvent = DocumentEvent.createEvent("MutationEvent");
mutationEvent.initEvent("DOMNodeInserted", true, false, relatedNode,
prevValue, newValue, attrName, attrChange);
// -->
</script>
</html>
```

MutationEvent.attrChange

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEvent.attrChange`

Description

The `attrChange` property of the `MutationEvent` object indicates the type of change that triggered the `DOMAttrModified` event. The values can be `MODIFICATION`, `ADDITION`, or `REMOVAL`.

Example

Listing 12.35 checks the `attrChange` property to determine how an attribute was modified and handles the event accordingly.

Listing 12.35 Examining the attrChange Property

```
<html>
<script language="JScript">
<!--
function handleMutation(evt) {
    if(evt.attrChange == MutationEvent.ADDITION)
        handleTextAreaAddition(evt);
// -->
</script>
</html>
```

MutationEvent.attrName

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEvent.attrName`

Description

The `attrName` property of the `MutationEvent` object indicates the name of the changed `Attr` node in a `DOMAttrModified` event.

Example

Listing 12.36 checks the `attrName` property to determine if the modified attribute is a `TextArea`. If so, the event is handled appropriately.

Listing 12.36 Examining the attrName Property

```
<html>
<script language="JScript">
<!--
function handleMutation(evt) {
    if(evt.attrName == "TextArea")
        handleTextAreaModification(evt);
// -->
</script>
</html>
```

MutationEvent.initMutationEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEventObj.initMutationEvent(type, canBubble, cancelable, relatedNode, prevValue, newValue, attrName, attrChange)`

Description

The `initMutationEvent()` method is used to initialize the value of a `MutationEvent` created using the `DocumentEvent` object. This method can only be called before the `MutationEvent` has been dispatched via the `dispatchEvent()` method, though it can be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

The possible values for `type` are `DOMSubTreeModified`, `DOMNodeInserted`, and `DOMNodeRemoved`.

Example

Listing 12.37 shows the creation of a mutation event object using the `createEvent()` of the `DocumentEvent` object. The event is then initialized using the `initEvent()` method to initialize it.

Listing 12.37 Creating and Initializing a MutationEvent

```
<html>
<script language="JScript">
<!--
var mutationEvent = DocumentEvent.createEvent("MutationEvent");
mutationEvent.initMutationEvent
    ("DOMNodeInserted", true, false, relatedNode,
    ↪prevValue, newValue, attrName, attrChange);
// -->
</script>
</html>
```

MutationEvent.newValue

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEventObj.newValue`

Description

The `newValue` property of the `MutationEvent` object indicates the new value of the `Attr` node in `DOMAttrModified` events and of the `CharacterData` node in `DOMCharDataModified` events.

Example

Listing 12.38 checks the `prevValue` and `newValue` properties to determine whether any actual changes have occurred to the `Attr` in question.

Listing 12.38 Comparing prevValue and newValue

```
<html>
<script language="JScript">
<!--
function handleMutation(evt) {
    if(evt.prevValue == evt.newValue)
        handleUnchangedEvent(evt);
// -->
</script>
</html>
```

MutationEvent.prevValue

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`mutationEventObj.prevValue`

Description

The `prevValue` property indicates the previous value of the `Attr` node in `DOMAttrModified` events, and of the `CharacterData` node in `DOMCharDataModified` events.

Example

Listing 12.39 checks the `prevValue` and `newValue` properties to determine whether any actual changes have occurred to the `Attr` in question.

Listing 12.39 Comparing prevValue and newValue

```
<html>
<script language="JScript">
<!--
function handleMutation(evt) {
    if(evt.prevValue == evt.newValue)
        handleUnchangedEvent(evt);
// -->
</script>
</html>
```

UIEvent

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

DOM Event object.

Description

The UIEvent object provides specific contextual information associated with User Interface events. The UIEvent object inherits all properties and methods from the Event object. Table 12.6 lists the properties and methods of the UIEvent object.

Table 12.6 Contents of the UIEvent Object

Type	Item	Description
Property	detail	Information regarding the event.
	view	Represents the abstract view from the UIEvent.
Method	initUIEvent()	Initializes a newly created UIEvent object.

Example

Listing 12.40 shows the creation of a user interface event object using the createEvent() method of the DocumentEvent object. The event is then initialized using the initUIEvent() method.

Listing 12.40 Creating and Initializing a User Interface Event Using the initUIEvent() Method of the UIEvent Object

```
<html>
<script language="JScript">
<!--
var uiEvent = DocumentEvent.createEvent("UIEvent");
uiEvent.initUIEvent("DOMFocusOut", true, false, );
// -->
</script>
</html>
```

UIEvent.detail

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`uiEventObj.detail`

Description

The `detail` property of the `UIEvent` object specifies some detail information about the Event, depending on the type of event.

Example

Listing 12.41 shows the creation of a user interface event object using the `createEvent()` method of the `DocumentEvent` object. The event is then initialized using the `initUIEvent()` method. After initialization, the `detail` property is written to.

Listing 12.41 *Writing the `detail` Property of the `UIEvent` Object*

```
<html>
<script language="JScript">
<!--
var uiEvent = DocumentEvent.createEvent("UIEvent");
uiEvent.initUIEvent("DOMFocusOut", true, false, );
uiEvent.detail =
// -->
</script>
</html>
```

UIEvent.initUIEvent()

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`uiEventObj.initUIEvent(type, canBubble, cancelable, view, detail)`

Description

The `initUIEvent()` method of the `UIEvent` object is used to initialize the value of an `UIEvent` created through the `DocumentEvent` object. This method can only be called before the `UIEvent` has been dispatched via the `dispatchEvent()` method, although it can be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

The `type` parameter is a string that can be *DOMFocusIn*, *DOMFocusOut*, or *DOMActivate*. The `canBubble` parameter indicates whether this event can bubble, the `cancelable` parameter indicates whether this event can be canceled, `view` is the `AbstractView` object in which this `UIEvent` originated, and `detail` is a number that

represents some type of detailed information regarding an UIEvent of the specified type.

Example

Listing 12.42 shows the creation and initialization of a UIEvent using the `initUIEvent()` method.

Listing 12.42 Creating and Initializing a User Interface Event

```
<html>
<script language="JScript">
<!--
var mouseEvent = DocumentEvent.createEvent("UIEvent");
mouseEvent.initUIEvent("DOMFocusOut", true, false, );
// -->
</script>
</html>
```

UIEvent.view

JavaScript 1.5+, JScript 5.0+
Nav6+, IE5+

Syntax

`uiEventObj.view`

Description

The `view` property of the `UIEvent` object identifies the `AbstractView` from which the event was generated.

Example

Listing 12.43 show the writing of the `view` property of the `UIEvent` object.

Listing 12.43 Creating and Initializing a User Interface Event

```
<html>
<script language="JScript">
<!--
var uiEvent = DocumentEvent.CreateEvent("UIEvent");
uiEvent.InitUIEvent("DOMFocusOut", true, false, );
uiEvent.view = "display";
// -->
</script>
</html>
```


CHAPTER 13

DOM CSS

Counter

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS object.

Description

Table 13.1 *Arguments Associated With Counter Object*

Type	Item	Description
Property	identifier	This is a read-only property of type <code>string</code> . Represents the identifier of the counter.
	listStyle	This read-only property of type <code>string</code> . Represents the style of the list.
	separator	This read-only property of type <code>string</code> . Represents the separator of the nested counter.

Example

Listing 13.1 illustrates reading the properties of the Counter object.

Listing 13.1 *Reading Properties of the Counter Object*

```
<html>
<script language="JScript">
<!--
var counterObj = primitiveValObj.getCounterValue();
var idVal = counterObj.identifier;
var listStyleVal = counterObj.listStyle;
var separatorVal = counterObj.separator;
// -->
</script>
</html>
```

Counter.identifier

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

CounterObj.identifier

Description

The read-only property identifier of the Counter object is used for to identify the counter.

Example

Listing 13.2 illustrates reading the identifier property of the Counter object.

Listing 13.2 *Reading the identifier property of the Counter Object*

```
<html>
<script language="JScript">
<!--
var counterObj = primitiveValObj.getCounterValue();
var idVal = counterObj.identifier;
var listStyleVal = counterObj.listStyle;
var separatorVal = counterObj.separator;
// -->
</script>
</html>
```

Counter.listStyle

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

counterObj.listStyle

Description

The read-only property `listStyle` of the `Counter` object is of type `String`. The `listStyle` property is used for the style of the list.

Example

Listing 13.3 illustrates reading the `listStyle` property of the `Counter` object.

Listing 13.3 Reading Properties of the Counter Object

```
<html>
<script language="JScript">
<!--
var counterObj = primitiveValObj.getCounterValue();
var idVal = counterObj.identifier;
var listStyleVal = counterObj.listStyle;
var separatorVal = counterObj.separator;
// -->
</script>
</html>
```

Counter.separator

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`Counter.separator`

Description

The read-only property `separator` of the `Counter` object is of type `String`. The `separator` property is used for the separator of the nested counters.

Example

Listing 13.4 illustrates reading the `separator` property of the `Counter` object.

Listing 13.4 Reading Properties of the Counter Object

```
<html>
<script language="JScript">
<!--
var counterObj = primitiveValObj.getCounterValue();
var idVal = counterObj.identifier;
var listStyleVal = counterObj.listStyle;
var separatorVal = counterObj.separator;
// -->
</script>
</html>
```

CSSCharsetRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSCharsetRule` object represents a `@charset` rule in a `CSSStyleSheet`. `CSSCharsetRule` extends `CSSRule`. The property associated with the `CSSCharsetRule` object is as follows:

<i>Item</i>	<i>Description</i>
encoding	This is a read-only property of type <code>String</code> . Represents a list of media types for this rule.

Example

Listing 13.5 demonstrates reading the encoding property of the `CSSCharsetRule` object.

Listing 13.5 Reading the encoding Property

```
<html>
<script language="JScript">
<!--
document.write(cssCharsetRuleObj);
// -->
</script>
</html>
```

CSSCharsetRule.encoding

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssCharsetRuleObj.encoding`

Description

The read-only property `encoding` of the `CSSCharsetRule` object is of type `String`. Encoding contains the encoding information used in this rule. This attribute can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR` and `SYNTAX_ERR`.

Example

Listing 13.6 demonstrates reading the encoding property of the `CSSCharsetRule` object.

Listing 13.6 *Reading the encoding Property*

```
<html>
<script language="JScript">
<!--
document.write(cssCharsetRuleObj.encoding);
// -->
</script>
</html>
```

CSSFontFaceRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSFontFaceRule` object represents a set of font descriptions in a `CSSStyleSheet`. `CSSFontFaceRule` inherits all properties and methods from `CSSRule`. The property associated with the `CSSFontFaceRule` object is as follows:

<i>Item</i>	<i>Description</i>
<code>style</code>	This is a read-only property of type <code>CSSStyleDeclaration</code> . Represents declaration block of the rule.

Example

Listing 13.7 demonstrates manipulating the `style` property of the `CSSFontFaceRule` object.

Listing 13.7 *Manipulating the style Property*

```
<html>
<script language="JScript">
<!--
cssFontFaceRuleObj.style.face = "courier";
// -->
</script>
</html>
```

CSSFontFaceRule.style

JavaScript 1.5+, JScript 5+
Nav6+, IE5+JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssFontFaceRuleObj.style`

Description

This read-only attribute is of type `CSSStyleDeclaration`. This attribute represents the declaration block of the rule.

Example

Listing 13.8 demonstrates manipulating the `style` property of the `CSSFontFaceRule` object.

Listing 13.8 Manipulating the style Property

```
<html>
<script language="JScript">
<!--
cssFontFaceRuleObj.style.face = "ariel";
// -->
</script>
</html>
```

CSSImportRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSImportRule` object represents the `@import` rule of a `CSSStyleSheet`. `CSSImportRule` inherits all methods and properties of `CSSRules`. Table 13.2 lists all properties of the `CSSImportRule` object.

Table 13.2 *Properties Associated with the CSSImportRule Object*

Item	Description
<code>href</code>	This is a read-only property of type <code>String</code> . Represents a textual representation of the location of the style sheet to be imported.
<code>media</code>	This is a read-only property of type <code>MediaList</code> . Represents a list of media types for which the style sheet may be used.
<code>styleSheet</code>	This is a read-only property of type <code>CSSStyleSheet</code> . Represents the style sheet referred to by this rule.

Example

Listing 13.9 demonstrates reading the `href` property of the `CSSImportRule` object.

Listing 13.9 *Reading the href Property*

```
<html>
<script language="JScript">
<!--
document.write(cssImportRuleObj.href);
// -->
</script>
</html>
```

CSSImportRule.href

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssImportRuleObj.href

Description

The read-only property href of the CSSImportRule object is of type string. The href property is the textual representation of the location of the style sheet to be imported. The URI specifier won't be included in this attribute.

Example

Listing 13.10 demonstrates reading the href property of the CSSImportRule object.

Listing 13.10 *Reading the href Property*

```
<html>
<script language="JScript">
<!--
document.write(cssImportRuleObj.href);
// -->
</script>
</html>
```

CSSImportRule.media

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssImportRuleObj.media

Description

The read-only property media of the CSSImportRule object is of type `MediaList`. The media property is a list of media types that can be used by the style sheet.

Example

Listing 13.11 demonstrates reading the media property of the CSSImportRule object.

Listing 13.11 *Reading the media Property*

```
<html>
<script language="JScript">
<!--
document.write(cssImportRuleObj.media);
// -->
</script>
</html>
```

CSSImportRule.styleSheet

JavaScript 1.5+, JScript 5+
Nav6+, IE5+JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssImportRuleObj.styleSheet

Description

The read-only property `styleSheet` of the `CSSImportRule` object is of type `CSSStyleSheet`. The `styleSheet` property represents the loaded style sheet of this rule. If a style sheet hasn't been loaded by this rule, the value will be `Null`.

Example

Listing 13.12 demonstrates reading the `styleSheet` property of the `CSSImportRule` object.

Listing 13.12 *Reading the styleSheet Property*

```
<html>
<script language="JScript">
<!--
document.write(cssImportRuleObj.styleSheet);
// -->
</script>
</html>
```

CSSMediaRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSMediaRule` object represents the target media types of a set of rules. `CSSMediaRule` extends `CSSRule`. Table 13.3 lists all methods and properties of the `CSSMediaRule` object.

Table 13.3 *Arguments Associated with the `CSSMediaRule` Object*

Type	Item	Description
Property	<code>cssRules</code>	This is a read-only property of type <code>CSSRuleList</code> . Represents a list of all rules within the media block.
	<code>media</code>	This is a read-only property of type <code>MediaList</code> . Represents a list of media types for this rule.
Method	<code>deleteRule()</code>	This method has no return value, but is used to delete a rule from the media block.
	<code>insertRule()</code>	This method returns a number representing the index within the rule collection of the newly inserted rule.

Example

Listing 13.13 gets a `CSSMediaRule` and sets its media type.

Listing 13.13 *Setting a Media Type*

```
<html>
<script language="JScript">
<!--
mediaRuleObj.media = "screen";
// -->
</script>
</html>
```

CSSMediaRule.cssRules

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssMediaRuleObj.cssRules`

Description

The `cssRule` property of the `CSSMediaRule` object is a read-only attribute of type `CSSRuleList`. This is a list of CSS rules contained within the media block.

Example

Listing 13.14 demonstrates iterating through the `cssRules` property and processing each item.

Listing 13.14 Iterating Through `cssRules`

```
<html>
<script language="JScript">
<!--
var ruleList = cssMediaRuleObj.cssRules;
var i = 0;
while( i < ruleList.length) {
    processRule(ruleList.item(i);
    i++;
}
// -->
</script>
</html>
```

CSSMediaRule.deleteRule()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssMediaRuleObj.deleteRule(index)`

Description

The `deleteRule()` method of the `CSSMediaRule` takes the argument *Number index*. This method deletes the rule from the media block. This method can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR` and `SYNTAX_ERR`.

Example

Listing 13.15 demonstrates removing a CSS media rule from the media block.

Listing 13.15 Removing a CSS Media Rule Using the `deleteRule()` **Method**

```
<html>
<script language="JScript">
<!--
cssMediaRule.deleteRule(1);
// -->
</script>
</html>
```

CSSMediaRule.insertRule()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssmediaRuleObj.insertRule(rule, index)`

Description

The `insertRule()` method of the `CSSMediaRule` object takes the arguments *String rule* and *Number index* and returns the index of the newly inserted rule. This method can raise a `DOMException` with the values of `HIERARCHY_REQUEST_ERR`, `INDEX_SIZE_ERR`, `NO_MODIFICATION_ALLOWED_ERR`, and `SYNTAX_ERR`.

Example

Listing 13.16 illustrates inserting a rule into the media block.

Listing 13.16 *Inserting a Rule into the Media Block Using the `insertRule()` Method*

```
<html>
<script language="JScript">
<!--
cssMediaRuleObj.insertRule("media:screen", 1);
// -->
</script>
</html>
```

CSSMediaRule.media

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssMediaRuleObj.media`

Description

The `media` property of the `CSSMediaRule` object is a read-only attribute of type `MediaList`. This is a list of media types for this rule.

Example

Listing 13.17 getting a `CSSMediaRule` and setting its media type.

Listing 13.17 *Setting a Media Type Using the `media` Property*

```
<html>
<script language="JScript">
<!--
mediaRuleObj.media = "handheld";
// -->
</script>
</html>
```

CSSPageRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The CSSPageRule object represents a @page rule within a CSSStyleSheet. The CSSPageRule extends CSSRule. Table 13.4 lists all properties of the CSSPageRule object.

Table 13.4 *Properties Associated with the CSSPageRule Object*

Item	Description
selectorText	This is a read-only property of type String. Represents a textual representation of the page selector of the rule.
style	This is a read-only property of type CSSStyleDeclaration. Represents a declaration block of this rule.

Example

Listing 13.18 demonstrates setting the selectorText property of a CSSPageRule object.

Listing 13.18 *Setting the selectorText Property*

```
<html>
<script language="JScript">
<!--
cssPageRuleObj.selectorText = "some selector text";
// -->
</script>
</html>
```

CSSPageRule.selectorText

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

CSSPageRule.selectorText

Description

The selectorText property of the CSSPageRule object is a read-only attribute of type String. The selectorText property is a textual representation of the page selector. This attribute can raise a DOMException with the values of NO_MODIFICATION_ALLOWED_ERR and SYNTAX_ERR.

Example

Listing 13.19 demonstrates setting the `selectorText` property of a `CSSPageRule` object.

Listing 13.19 Setting the `selectorText` Property

```
<html>
<script language="JScript">
<!--
cssPageRuleObj.selectorText = "selection";
// -->
</script>
</html>
```

CSSPageRule.style

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssPageRuleObj.style`

Description

The `style` property of the `CSSPageRule` object is a read-only attribute of type `CSSStyleDeclaration`. The `style` property represents the declaration block of this rule.

Example

Listing 13.20 demonstrates manipulating the `style` property of a `CSSPageRule` object.

Listing 13.20 Setting a Property of the `style` Property of the `CSSPageRule` Object

```
<html>
<script language="JScript">
<!--
cssPageRuleObj.style.size = "10";
// -->
</script>
</html>
```

CSSPrimitiveValue

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSPrimitiveValue` object represents a `CSSValue`. This object can be used to determine or set a value for a specific style property. Table 13.5 lists all constants, properties, and methods of the `CSSPrimitiveValue` object.

Table 13.5 *Arguments Associated with the `CSSPrimitiveValue` Object*

Type	Item	Description
Constant	CSS_ATTR	This is a constant of type <code>Number</code> . The value is 22, which represents an attribute function.
	CSS_CM	This is a constant of type <code>Number</code> . The value is 6, which represents a length.
	CSS_COUNTER	This is a constant of type <code>String</code> . The value is 23, which represents a counter or counter function.
	CSS_DEG	This is a constant of type <code>Number</code> . The value is 11, which represents an angle.
	CSS_DIMENSION	This is a constant of type <code>Number</code> . The value is 18, which represents a dimension.
	CSS_EMS	This is a constant of type <code>Number</code> . The value is 3, which represents a length.
	CSS_EXS	This is a constant of type <code>Number</code> . The value is 4, which represents a length.
	CSS_GRAD	This is a constant of type <code>Number</code> . The value is 13, which represents an angle.
	CSS_HZ	This is a constant of type <code>Number</code> . The value is 16, which represents a frequency.
	CSS_IDENT	This is a constant of type <code>Number</code> . The value is 21, which represents an identifier.
	CSS_IN	This is a constant of type <code>Number</code> . The value is 8, which represents a length.
	CSS_KHZ	This is a constant of type <code>Number</code> . The value is 17, which represents a frequency.
	CSS_MM	This is a constant of type <code>Number</code> . The value is 7, which represents a length.
	CSS_MS	This is a constant of type <code>Number</code> . The value is 14, which represents a time.
	CSS_NUMBER	This is a constant of type <code>Number</code> . The value is 1, which represents a simple number.
	CSS_PC	This is a constant of type <code>Number</code> . The value is 10, which represents a length.
	CSS_PERCENTAGE	This is a constant of type <code>Number</code> . The value is 2, which represents a percentage.
	CSS_PT	This is a constant of type <code>Number</code> . The value is 9, which represents a length.

Type	Item	Description
Property	CSS_PX	This is a constant of type <code>Number</code> . The value is 5, which represents a length.
	CSS_RAD	This is a constant of type <code>Number</code> . The value is 12, which represents an angle.
	CSS_RECT	This is a constant of type <code>Number</code> . The value is 24, which represents a <code>Rect</code> function.
	CSS_RGBCOLOR	This is a constant of type <code>Number</code> . The value is 25, which represents a RGB color.
	CSS_S	This is a constant of type <code>Number</code> . The value is 15, which represents a time.
	CSS_STRING	This is a constant of type <code>Number</code> . The value is 19, which represents a <code>String</code> .
	CSS_UNKNOWN	This is a constant of type <code>Number</code> . The value is 0, which represents a CSS2 value that isn't recognized.
	CSS_URI	This is a constant of type <code>Number</code> . The value is 20, which represents a URI.
	<code>primitiveType</code>	This is a read-only property of type <code>Number</code> . Represents a type of value defined by one of the constants.
Method	<code>getCounterValue()</code>	This method returns a <code>Counter</code> object.
	<code>getFloatValue()</code>	This method returns a number representing the float value in a specified unit.
	<code>getRectValue()</code>	This method returns a <code>Rect</code> object.
	<code>getRGBColorValue()</code>	This method returns a <code>RGBColor</code> .
	<code>getStringValue()</code>	This method returns a <code>String</code> representing a string value of the specified unit.
	<code>setFloatValue()</code>	This method has no return value. Sets the float value for a specified unit.
	<code>setStringValue()</code>	This method has no return value. Sets the string value for a specified unit.

Example

Listing 13.21 illustrates using the `getRGBColor()` method of the `CSSPrimitiveValue` object.

Listing 13.21 Using the `getRGBColor()` Method

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColor();
```

Listing 13.21 Continued

```
// -->
</script>
</html>
```

CSSPrimitiveValue.getCounterValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.getCounterValue()
```

Description

The `getCounterValue()` method of the `CSSPrimitiveValue` takes no arguments and returns a `Counter` object representing the Counter value. This method can raise a `DOMException` with the value of `INVALID_ACCESS_ERR`.

Example

Listing 13.22 Getting a counter value for the `CSSPrimitiveValue` object.

Listing 13.22 Reading the Counter Value Using the `getCounterValue()` Method

```
<html>
<script language="JScript">
<!--
var counterObj = cssPrimitiveValue.getCounterValue();
// -->
</script>
</html>
```

CSSPrimitiveValue.getFloatValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.getFloatValue(unitType)
```

Description

The `getFloatValue()` method of the `CSSPrimitiveValue` takes the argument `unitType` and returns a float value. This method can raise a `DOMException` with the value of `INVALID_ACCESS_ERR`.

Example

Listing 13.23 demonstrates the assigning of an event target and fires off an event.

Listing 13.23 *Getting a Float Value Using the `getFloatValue()` Method of `CSSPrimitiveValue` Object*

```
<html>
<script language="JScript">
<!--
var floatVal = cssPrimitiveValue.getFloatValue
    (CSSPrimitiveValue.CSS_PERCENTAGE);
// -->
</script>
</html>
```

CSSPrimitiveValue.getRectValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.getRectValue()
```

Description

The `getRectValue()` method of the `CSSPrimitiveValue` takes no arguments and returns a `Rect` object representing a `Rect` value. This method can raise a `DOMException` with the value of `INVALID_ACCESS_ERR`.

Example

Listing 13.24 Illustrates getting a `Rect` value for the `CSSPrimitiveValue` object.

Listing 13.24 *Getting a Rect Value Using the `getRectValue()` Method of the `CSSPrimitiveValue` Object*

```
<html>
<script language="JScript">
<!--
var rectObj = cssPrimitiveValue.getRectValue();
// -->
</script>
</html>
```

CSSPrimitiveValue.getRGBColorValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.getRGBColorValue()
```

Description

The `getRGBColorValue()` method of the `CSSPrimitiveValue` takes no arguments and returns a `RGBColor` object representing the RGB Color value. This method can raise a `DOMException` with the value of `INVALID_ACCESS_ERR`.

Example

Listing 13.25 illustrates using the `getRGBColorValue()` method of the `CSSPrimitiveValue` object.

Listing 13.25 Using the `getRGBColorValue()` Method

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColorValue();
// -->
</script>
</html>
```

CSSPrimitiveValue.getStringValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.getStringValue()
```

Description

The `getStringValue()` method of the `CSSPrimitiveValue` object takes no arguments and returns a `String` value in the current unit. This method can raise a `DOMException` with the value of `INVALID_ACCESS_ERR`.

Example

Listing 13.26 Getting a string value for the `CSSPrimitiveValue` object.

Listing 13.26 Getting a String Value Using the `getStringValue()` Method of `CSSPrimitiveValue`

```
<html>
<script language="JScript">
<!--
var stringObj = cssPrimitiveValye.getStringValue();
// -->
</script>
</html>
```

CSSPrimitiveValue.primitiveType

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssPrimitiveValueObj.primitiveType`

Description

The `primitiveType` property of the `CSSPrimitiveValue` object is a read-only attribute of type `Number`. The `primitiveType` property is a `Number` representing the type of value as defined by the constants.

Example

Listing 13.27 demonstrates reading the `primitiveType` property of the `CSSPrimitiveValue` object.

Listing 13.27 Accessing the `primitiveType` Property

```
<html>
<script language="JScript">
<!--
if(cssPrimitiveValObj.primitiveType == CSSPrimitiveValue.CSS_STRING)
    processStringType(cssPrimitiveValObj.primitiveType);
// -->
</script>
</html>
```

CSSPrimitiveValue.setFloatValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssPrimitiveValueObj.setFloatValue(unitType, floatType)`

Description

The `setFloatValue()` method of the `CSSPrimitiveValue` takes the arguments `unitType` and `floatType` and has no return value. This method can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR` and `INVALID_ACCESS_ERR`.

Example

Listing 13.28 demonstrates setting a float value for `CSSPrimitiveValue`.

Listing 13.28 Setting a float Value Using the setFloatValue() Method of the CSSPrimitiveValue Object

```
<html>
<script language="JScript">
<!--
cssPrimitiveValue.setFloatValue(CSSPrimitiveValue.CSS_PERCENTAGE, 3.34);
// -->
</script>
</html>
```

CSSPrimitiveValue.setStringValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssPrimitiveValueObj.setStringValue(stringType, stringValue)
```

Description

The `setStringValue()` method of the `CSSPrimitiveValue` takes the arguments `stringType` and `stringValue` and has no return value. This method can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR` and `INVALID_ACCESS_ERR`.

Example

Listing 13.29 setting a string value for the `CSSPrimitiveValue` object.

Listing 13.29 Setting a String Value Using the setStringValue() Method of the CSSPrimitiveValue Object

```
<html>
<script language="JScript">
<!--
cssPrimitiveValue.setStringValue
    (CSSPrimitiveValue.CSS_STRING, "a random string");
// -->
</script>
</html>
```

CSSRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

CSSRule is an abstract base object for any CSS statement including the rules set and at rules. Table 13.6 lists all the properties of the CSSRule object.

Table 13.6 *Properties Associated with the CSSRule Object*

Item	Description
cssText	This is a read-only property of type String that returns a textual representation of the current state of the rule.
parentRule	This is a read-only property of type CSSRule that returns the containing rule or Null.
parentStyleSheet	This is a read-only property of type CSSStyleSheet that returns the StyleSheet that contains this rule.
type	This is a read-only property that returns a number indicating the type of rule.

Example

Listing 13.30 illustrates getting a CSSRules object and iterating through the list while checking each rule's type.

Listing 13.30 *Checking Rule Type*

```
<html>
<script language="JScript">
<!--
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    var ruleObj = ruleList.index(i);
    if(ruleObj.type == UNKNOWN_RULE)
        handleUnknownRule(ruleObj);
    i++;
}
// -->
</script>
</html>
```

CSSRule.cssText

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssRuleObj.cssText

Description

The `cssText` property of the `CSSRule` object returns a string. This is a textual representation of the CSS rule. This String holds the current state of the CSS rule. This attribute can raise a `DOMException` with the values of `HIERARCHY_REQUEST_ERR`, `NO_MODIFICATION_ALLOWED_ERR`, `INVALID_MODIFICATION_ERR`, and `SYNTAX_ERR`.

Example

Listing 13.31 illustrates the setting of the `cssText` property for a `CSSRule` object.

Listing 13.31 *Setting CSS Text Using the `cssText` Property of the `CSSRule` Object*

```
<html>
<script language="JScript">
<!--
cssRule.cssText = "color:'green'; font-weight:bold";
// -->
</script>
</html>
```

CSSRule.parentRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssRuleObj.parentRule`

Description

The read-only property `parentRule` of the `CSSRule` object is of type `CSSRule`. If this `CSSRule` is contained within another `CSSRule`, this is the containing rule. Otherwise this is `Null`.

Example

Listing 13.32 demonstrates reading the `parentRule` property.

Listing 13.32 *Reading the `parentRule` Property*

```
<html>
<script language="JScript">
<!--
if(cssRuleObj.parentRule.name = "myfontrule")
    processRule(cssRuleObj.parentRule);
// -->
</script>
</html>
```

CSSRule.parentStyleSheet

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssRuleObj.parentStyleSheet`

Description

The read-only property `parentStyleSheet` of the `CSSRule` object is of type `CSSStyleSheet`. Represents the style sheet that contains this rule.

Example

Listing 13.33 demonstrates reading the `parentStyleSheet` property.

Listing 13.33 *Reading the `parentStyleSheet` Property*

```
<html>
<script language="JScript">
<!--
if(cssRuleObj.parentStyleSheet.name = "mycss")
    processMyCSS(cssRuleObj.parentStyleSheet);
// -->
</script>
</html>
```

CSSRule.type

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssRuleObj.type`

Description

The read-only property `type` of the `CSSRule` object returns a number indicating the type of rule.

Example

Listing 13.34 getting a `CSSRules` object and iterating through the list while checking each rule's type.

Listing 13.34 *Checking Rule Type Reading the `type` Property of `CSSRule`*

```
<html>
<script language="JScript">
<!--
```

Listing 13.34 Continued

```
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    var ruleObj = ruleList.item(i);
    if(ruleObj.type == MEDIA_RULE)
        handleUnknownRule(ruleObj);
    i++;
}
// -->
</script>
</html>
```

CSSRuleList

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

CSSRuleList is an object that acts as an abstraction of an ordered collection of CSS Rules. Table 13.7 lists all properties and methods of the CSSRuleList object.

Table 13.7 Arguments Associated with the CSSRulesList Object

Type	Item	Description
Property	length	This is a read-only property that returns the number of CSS rules in the list.
Method	item()	This method returns the CSS rule by the ordinal index.

Example

Listing 13.35 getting a CSSRuleList object and iterating through the list.

Listing 13.35 Iterating Through a CSSRuleList Object

```
<html>
<script language="JScript">
<!--
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    processRule(ruleList.item(i);
    i++;
}
// -->
</script>
</html>
```

CSSRuleList.item()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssRuleListObj.item(index)`

Description

The `item()` method of the `CSSRuleList` takes the argument *index*. This method is used to retrieve the CSS rule by the ordinal index.

Example

Listing 13.36 getting a `CSSRuleList` object and iterating through the list.

Listing 13.36 Iterating Through a CSSRuleList Using the item() Method

```
<html>
<script language="JScript">
<!--
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    processRule(ruleList.item(i));
    i++;
}
// -->
</script>
</html>
```

CSSRuleList.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssRuleListObj.length`

Description

The `length` property of the `CSSRuleList` object returns the number of CSS rules contained within the `CSSRuleList`.

Example

Listing 13.37 getting a `CSSRuleList` object and iterating through the list.

Listing 13.37 *Reading the length Property of the CSSRuleList Object*

```
<html>
<script language="JScript">
<!--
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    processRule(ruleList.item(i);
    i++;
}
// -->
</script>
</html>
```

CSSStyleDeclaration

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSStyleDeclaration` object represents a CSS declaration block. `CSSStyleDeclaration` can be used to set or determine what style properties are set within the block. Table 13.8 lists all properties and methods of the `CSSStyleDeclaration` object.

Table 13.8 *Arguments Associated with the CSSMediaRule Object*

Type	Item	Description
Property	<code>cssText</code>	This is a property of type <code>string</code> . Represents a textual representation of the declaration block.
	<code>length</code>	This read-only property of type <code>Number</code> . Represents the number of properties that have been set in the declaration block.
	<code>parentRule</code>	This read-only property of type <code>CSSRule</code> . Represents the <code>CSSRule</code> that contains this declaration block.
Method	<code>item()</code>	This method returns a <code>string</code> representing the property at this ordinal position.
	<code>getPropertyCSSValue()</code>	This method returns a <code>cssValue</code> representing the object representation of the value of the CSS property.
	<code>getPropertyPriority()</code>	This method returns a <code>string</code> representing the priority of the CSS property.

Type	Item	Description
	<code>getPropertyValue()</code>	This method returns a <code>String</code> representing the value of a CSS property.
	<code>removeProperty()</code>	This method returns a <code>String</code> representing the property that has been set within the declaration block.
	<code>setProperty()</code>	This method has no return value, but is used to set the property value and priority.

Example

Listing 13.38 demonstrates setting the `CSSText` property of the `CSSStyleDeclaration` object.

Listing 13.38 Setting the `CSSText` Property

```
<html>
<script language="JavaScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.cssText = "color:'green'; font-weight:bold";
// -->
</script>
</html>
```

CSSStyleDeclaration.cssText

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssStyleDeclObj.cssText
```

Description

The `cssText` property of the `CSSStyleDeclaration` object is a textual representation of the declaration block. Setting this attribute will reset all properties set in this declaration block. This attribute can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR` and `SYNTAX_ERR`.

Example

Listing 13.39 demonstrates setting the `cssText` property of `CSSStyleDeclaration` object.

Listing 13.39 Setting the `cssText` Property

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.cssText = "color:'red'; font-weight:bold";
// -->
</script>
</html>
```

CSSStyleDeclaration.getPropertyCSSValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStuleDeclObj.getPropertyCSSValue(propertyName)`

Description

The `getPropertyCSSValue()` method of the `CSSStyleDeclaration` takes the argument *string propertyName* and returns a `CSSValue` representing the value of the property of the property that has been set. Returns Null if the property hasn't been set.

Example

Listing 13.40 illustrates getting a property value using the `getPropertyCSSValue()` method of `CSSStyleDeclaration`.

Listing 13.40 Invoking the `getPropertyCSSValue()` Method

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.getPropertyCSSValue("height");
// -->
</script>
</html>
```

CSSStyleDeclaration.getPropertyPriority()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStyleDecl.getPropertyPriority(propertyName)`

Description

The `getPropertyPriority()` method of the `CSSStyleDeclaration` takes the argument *propertyName* and returns a `String` representing the priority of the CSS property. Returns an empty string if none exists.

Example

Listing 13.41 illustrates getting a property value using the `getPropertyPriority()` method of `CSSStyleDeclaration`.

Listing 13.41 *Getting the Property Priority Using the `getPropertyPriority()` Method*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.getPropertyPriority("height");
// -->
</script>
</html>
```

CSSStyleDeclaration.getPropertyValue()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssStyleDeclObj.getPropertyValue(propertyName)
```

Description

The `getPropertyValue()` method of the `CSSStyleDeclaration` takes the argument *propertyName* and returns a `String` representing the value of the property in the declaration block. Returns an empty string if the property hasn't been set.

Example

Listing 13.42 illustrates getting a property value using the `getPropertyValue()` method of `CSSStyleDeclaration`.

Listing 13.42 *Getting a Named Property Value Using the `getPropertyValue()` Method*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.getPropertyValue("height");
```

Listing 13.42 Continued

```
// -->
</script>
</html>
```

CSSStyleDeclaration.item()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStyleDeclarationObj.item(index)`

Description

The `item()` method of the `CSSStyleDeclaration` takes the argument *index* and returns a `String` representing the name of the property at the ordinal position.

Example

Listing 13.43 illustrates getting a `CSSValue` and setting its value type.

Listing 13.43 Invoking the `item()` Method

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
var cssValueObj = cssStyleDeclarationObj.getPropertyCSSValue("height");
document.write(cssValueObj.cssValueType);
// -->
</script>
</html>
```

CSSStyleDeclaration.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStyleDeclObj.length`

Description

The `length` property of the `CSSStyleDeclaration` object is a read-only attribute of type `Number`. `Length` is the number of properties that have been set in this declaration block.

Example

Listing 13.44 illustrates reading the `length` property of `CSSStyleDeclaration`.

Listing 13.44 *Getting the length Property*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
document.write(cssStyleDeclarationObj.length);
// -->
</script>
</html>
```

CSSStyleDeclaration.parentRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssStyleDeclObj.parentRule

Description

The `parentRule` read-only property of the `CSSStyleDeclaration` object. The `parentRule` property represents the `CSSRule` that contains this declaration block or is Null if this block isn't contained in a `CSSRule`.

Example

Listing 13.45 illustrates reading the `parentRule` property of `CSSStyleDeclaration`.

Listing 13.45 *Setting the parentRule Property*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
document.write(cssStyleDeclarationObj.parentRule);
// -->
</script>
</html>
```

CSSStyleDeclaration.removeProperty()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssStyleDeclObj.removeProperty(propertyName)

Description

The `removeProperty()` method of the `CSSStyleDeclaration` takes the argument *propertyName* and returns a String representing the value of the property that has

been set. Returns an empty string if the property hasn't been set. This method can raise a `DOMException` with the value of `NO_MODIFICATION_ALLOWED_ERR`.

Example

Listing 13.46 illustrates getting a property value using the `removeProperty()` method of `CSSStyleDeclaration`.

Listing 13.46 Removing a Property Using the `removeProperty()` Method of the `CSSStyleDeclaration` Object

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
cssStyleDeclarationObj.removeProperty("height");
// -->
</script>
</html>
```

CSSStyleDeclaration.setProperty()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssStyleDeclObj.setProperty(propertyName, value, priority)
```

Description

The `setProperty()` method of the `CSSStyleDeclaration` takes the arguments *string propertyName*, *string value*, *string priority*, and has no return value. This method can raise a `DOMException` with the value of `NO_MODIFICATION_ALLOWED_ERR`.

Example

Listing 13.47 illustrates setting a property value using the `setProperty()` method of `CSSStyleDeclaration`.

Listing 13.47 Setting a Property Value Using the `setProperty()` Method of `CSSStyleDeclaration` Object

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.Item(0);
cssStyleDeclarationObj.setProperty("height", "35");
// -->
</script>
</html>
```

CSSStyleRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

CSSStyleRule is an object representing a single rule set in a CSSStyleSheet. CSSStyleRule extends CSSRule. Table 13.9 lists all properties of the CSSStyleRule object.

Table 13.9 *Properties Associated with the CSSRule Object*

Item	Description
selectorText	This is a property of type string. Represents a textual representation of the selector for this rule set.
style	This is a read-only property of type CSSStyleDeclaration. Represents the declaration block of the rule set.

Example

Listing 13.48 gets a CSSStyleRule object and sets its selectorText property.

Listing 13.48 *Assigning and Sending an Event*

```
<html>
<script language="JScript">
<!--
var cssStyleRuleObj = bodObj.style;
cssStyleRuleObj.selectorText = "some text";
// -->
</script>
</html>
```

CSSStyleRule.selectorText

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

cssStyleRuleObj.selectorText

Description

The selectorText of the CSSStyleRule object is a read-only attribute of type string. This is a textual representation of the selector of this rule set. This attribute can raise a DOMException with the values of `NO_MODIFICATION_ALLOWED_ERR` and `SYNTAX_ERR`.

Example

Listing 13.49 getting a `CSSStyleRule` object and setting its `selectorText` property.

Listing 13.49 *Setting* `selectorText`

```
<html>
<script language="JScript">
<!--
var cssStyleRuleObj = bodObj.style;
cssStyleRuleObj.selectorText = "selector text";
// -->
</script>
</html>
```

CSSStyleRule.style

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStyleRuleObj.style`

Description

The `style` property of the `CSSStyleRule` object is a read-only property of type `CSSStyleDeclaration`. This contains the declarative block of the rule set.

Example

Listing 13.50 demonstrates the assignment of a value to the `height` property of the `style` property of the `CSSStyleRule` object.

Listing 13.50 *Writing the* `style` *Property of the* `CSSStyleRule` *Object*

```
<html>
<script language="JScript">
<!--
cssStyleRuleObj.style.height = 24;
// -->
</script>
</html>
```

CSSStyleSheet

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSStyleSheet` object is used to represent a CSS style sheet. This object allows access to the collection of rules within a CSS StyleSheet. Table 13.10 lists all properties and methods of the `CSSStyleSheet` object.

Table 13.10 Arguments Associated with the `CSSStyleSheet` Object

Type	Item	Description
Property	<code>cssRule</code>	This is a read-only property that is a <code>CSSRuleList</code> object.
	<code>ownerRule</code>	This is a read-only property that is a <code>CSSRule</code> object.
Method	<code>deleteRule(index)</code>	This method is used to delete a rule from a style sheet.
	<code>insertRule()</code>	This method is used to insert a new rule into a style sheet.

Example

Listing 13.51 demonstrates the creation of a `CSSStyleSheetObject`.

Listing 13.51 Creating a `CSSStyleSheet` Object

```
<html>
<script language="JScript">
<!--
var cssStyleSheetObj = DOMImplementationCSS.createCSSStyleSheet
    ("mycss", "handheld");
// -->
</script>
</html>
```

`CSSStyleSheet.cssRules`

JavaScript 1.5+, JScript 5+

Nav6+, IE5+

Syntax

`cssStyleSheetObj.cssRules`

Description

The `cssRules` property of the `CSSStyleSheet` object is a read-only property that is of type `CSSRuleList` object. `cssRules` is a list of all rules contained within the style sheet and includes both *rule sets* and *at-rules*.

Example

Listing 13.52 demonstrates iterating through the list of `cssRules` and displaying the type of each.

Listing 13.52 Iterating Through `cssRules`

```
<html>
<script language="JScript">
<!--
var rulesList = cssStyleSheetObj.cssRules;
var i = 0;
while(i < ruleList.length) {
    display(ruleList.Index(i).type);
    i++;
}
// -->
</script>
</html>
```

CSSStyleSheet.deleteRule()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssStyleSheetObj.deleteRule(index)
```

Description

The `deleteRule()` method of the `CSSStyleSheet` takes the argument *index*. This method is used to remove a rule from the style sheet at position *index*. This method can raise a `DOMException` with the values of `INDEX_SIZE_ERR` and `NO_MODIFICATION_ALLOWED_ERR`.

Example

Listing 13.53 demonstrates removing a `CSSRule` from the stylesheet.

Listing 13.53 Removing a Rule Using the `deleteRule()` Method of the `CSSStyleSheet` Object

```
<html>
<script language="JScript">
<!--
cssStyleSheetObj.deleteRule(0);
// -->
</script>
</html>
```

CSSStyleSheet.insertRule()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssStyleSheetObj.insertRule(rule, index)
```


Description

The `insertRule()` method of the `CSSStyleSheet` takes the arguments *rule* and *index* and returns the index of the newly inserted rule. This method can raise a `DOMException` object with the values of `HIERARCHY_REQUEST_ERR`, `INDEX_SIZE_ERR`, `NO_MODIFICATION_ALLOWED_ERR`, and `SYNTAX_ERR`.

Example

Listing 13.54 illustrates inserting a new `CSSRule` into the style sheet.

Listing 13.54 *Inserting a `CSSRule` Using the `insertRule()` Method of the `CSSStyleSheet` Object*

```
<html>
<script language="JScript">
<!--
cssStyleSheetObj.insertRule("margin-top: 1ex", 0);
// -->
</script>
</html>
```

CSSStyleSheet.ownerRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssStyleSheetObj.ownerRule`

Description

This is a read-only attribute that is of type `CSSRule` object. `ownerRule` can contain both *rule sets* and *at-rules*.

Example

Listing 13.55 demonstrates reading the `ownerRule` property.

Listing 13.55 *Reading `ownerRule`*

```
<html>
<script language="JScript">
<!--
if(cssStyleSheetObj.ownerRule.type = "Media") {
    handleMediaRuler(cssStyleSheetObj.ownerRule);
// -->
</script>
</html>
```

CSSUnknownRule

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The CSSUnknownRule object represents an at-rule that isn't supported. It inherits all methods and properties from CSSRule.

Example

Listing 13.56 getting a CSSRules object and iterating through the list while checking to see whether the rule type is a CSSUnknownRule.

Listing 13.56 *Checking the type Property of the CSSUnknownRule Object*

```
<html>
<script language="JScript">
<!--
var ruleList = cssStyleSheetObj.rules;
var i = 0;
while( i < ruleList.length) {
    var ruleObj = ruleList.item(i);
    if(ruleObj.Type == UNKNOWN_RULE)
        handleUnknownRule(ruleObj);
    i++;
}
// -->
</script>
</html>
```

CSSValue

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The CSSValue object represents a simple or complex value within the context of a CSS property. Table 13.11 lists all constants and properties of the CSSValue object.

Table 13.11 Arguments Associated with the CSSValue Object

Type	Item	Description
Constants	CSS_CUSTOM	This is a constant of type Number. The value is 3, which represents that it is a custom value.
	CSS_INHERIT	This is a constant of type Number. The value is 0, which represents that the value is inherited.
	CSS_PRIMITIVE_VALUE	This is a constant of type String. The value is 1, which represents a primitive value and an instance of CSSPrimitiveValue.
	CSS_VALUE_LIST	This is a constant of type Number. The value is 2, which represents a CSSValueList and is an instance of CSSValueList.
Properties	cssText	This is a property of type String. It is a textual representation of the current value.
	cssValueType	This read-only property is of type Number. Represents a code describing the type of value.

Example

Listing 13.57 illustrates getting a CSSValue and setting its value type.

Listing 13.57 Getting a Property

```

<html>
<script language="JavaScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
var cssValueObj = cssStyleDeclarationObj.getPropertyCSSValue("height");
document.write(cssValueObj.cssValueType);
// -->
</script>
</html>

```

CSSValue.cssText

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssValueObj.cssText
```

Description

The `cssText` property of the `CSSValue` object is of type string. The `cssText` property is a textual representation of the current value. This attribute can raise a `DOMException` with the values of `NO_MODIFICATION_ALLOWED_ERR`, `INVALID_MODIFICATION_ERROR`, and `SYNTAX_ERR`.

Example

Listing 13.58 illustrates getting a `CSSValue` and setting its value type.

Listing 13.58 *Reading the `cssText` Property of the `CSSValue` Object*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
var cssValueObj = cssStyleDeclarationObj.getPropertyCSSValue("height");
document.write(cssValueObj.cssText);
// -->
</script>
</html>
```

CSSValue.cssValueType

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssValueObj.cssValueType`

Description

The read-only property `cssValueType` of the `CSSValue` object is of type `Number`. The `cssValueType` property is a code defining the type of value.

Example

Listing 13.59 illustrates getting a `CSSValue` and setting its value type.

Listing 13.59 *Reading the `cssValueType` Property of the `CSSValue` Object*

```
<html>
<script language="JScript">
<!--
var cssStyleDeclarationObj = cssStyleSheetObj.cssRules.item(0);
var cssValueObj = cssStyleDeclarationObj.getPropertyCSSValue("height");
document.write(cssValueObj.cssValueType);
// -->
</script>
</html>
```

CSSValueList

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `CSSValueList` object is an abstraction of an ordered collection of `CSSValues`. Table 13.12 lists the property and method of the `CSSValue` object.

Table 13.12 Arguments Associated with the `CSSValueList` Object

Type	Item	Description
Property	<code>length</code>	This is a read-only property of type <code>Number</code> . It represents the number of <code>CSSValues</code> in the list.
Method	<code>item()</code>	This method returns a <code>CSSValue</code> .

Example

Listing 13.60 illustrates iterating through a `CSSValueList` object.

Listing 13.60 Iterating Through a `CSSValueList`

```
<html>
<script language="JScript">
<!--
var i = 0;

while(i < valueListObj.length) {

    var listItem = valueListObj.item(i);
    processListItem(listItem);
    i++;
}
// -->
</script>
</html>
```

`CSSValueList.item()`

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
cssValueListObj.item(index)
```

Description

The `item()` method of the `CSSValueList` takes the argument *Number index* and returns the `CSSValue` at the index position in the list.

Example

Listing 13.61 illustrates iterating through a `CSSValueList` object.

Listing 13.61 Invoking the `item()` Method of the `CSSValueList` Object

```
<html>
<script language="JScript">
<!--
var i = 0;

while(i < valueListObj.length) {
    var listItem = valueListObj.item(i);
    processListItem(listItem);
    i++;
}
// -->
</script>
</html>
```

CSSValueList.length

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`cssValueListObj.length`

Description

This read-only attribute is of type `Number`. `Length` is a number representing the number of `CSSValue` objects in the list.

Example

Listing 13.62 illustrates iterating through a `CSSValueList` object.

Listing 13.62 Reading the `length` Property of the `CSSValueList` Object

```
<html>
<script language="JScript">
<!--
var i = 0;

while(i < valueListObj.length) {
    var listItem = valueListObj.item(i);
    processListItem(listItem);
    i++;
}
// -->
</script>
</html>
```

DocumentCSS

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The DocumentCSS object represents a document with a CSS view. The method associated with the DocumentCSS object is as follows:

<i>Item</i>	<i>Description</i>
<code>getOverrideStyle()</code>	This method returns a <code>CSSStyleSheet</code> representing the override style declaration.

Example

Listing 13.63 demonstrates getting the overridden style of a particular element.

Listing 13.63 Overriding the Style

```
<html>
<script language="JScript">
<!--
documentCSSObj.getOverrideStyle(bodyObj, null);
// -->
</script>
</html>
```

DocumentCSS.getOverrideStyle()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`documentCSSObj.getOverrideStyle(elt, pseudoElt)`

Description

The `getOverrideStyle()` method of the DocumentCSS takes the arguments *Element elt* and *String pseudoElt* and returns the override style declaration.

Example

Listing 13.64 demonstrates getting the overridden style of a particular element.

Listing 13.64 *Overriding the Style Using the `getOverrideStyle()` Method of the `DocumentCSS` Object*

```
<html>
<script language="JScript">
<!--
documentCSSObj.getOverrideStyle(tableObj, null);
// -->
</script>
</html>
```

DOMImplementationCSS

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `DOMImplementationCSS` object represents the ability for the DOM user to create a `CSSStyleSheet` outside the context of the document. The method associated with the `DOMImplementationCSS` object is as follows:

<i>Item</i>	<i>Description</i>
<code>createCSSStyleSheet</code>	This method returns a <code>CSSStyleSheet</code> object representing the new <code>CSSStyleSheet</code> .

Example

Listing 13.65 demonstrates creating a new style sheet.

Listing 13.65 *Creating a New Style Sheet*

```
<html>
<script language="JScript">
<!--
var cssStyleSheetObj = DOMImplementation.createCSSStyleSheet("my css",
➤ "print");
// -->
</script>
</html>
```

DOMImplementationCSS.createCSSStyleSheet()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`DOMImplementationCSS.createCSSStyleSheet(title, media)`

Description

The `CreateCSSStyleSheet()` method of the `DOMImplementationCSS` takes the arguments *String title* and *String media* and returns a new `CSSStyleSheet`. This method can raise a `DOMException` with the value of `SYNTAX_ERR`.

Example

Listing 13.66 demonstrates creating a new style sheet.

Listing 13.66 *Creating a New Style Sheet Using the `createCSSStyleSheet()` Method of the `DOMImplementationCSS` Object*

```
<html>
<script language="JScript">
<!--
var cssStyleSheetObj = DOMImplementation.createCSSStyleSheet("my css",
    "print");
// -->
</script>
</html>
```

ElementCSSInlineStyle

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The `ElementCSSInlineStyle` object represents the contents of the style attribute for HTML elements. The property associated with the `ElementCSSInlineStyle` object is as follows:

<i>Item</i>	<i>Description</i>
<code>style</code>	This is a read-only property of type <code>CSSStyleDeclaration</code> .

Example

Listing 13.67 demonstrates reading the style property from the `ElementCSSInlineStyle` object.

Listing 13.67 *Reading the style Property*

```
<html>
<script language="JScript">
<!--
var cssStyleDef = elementCSSInlineStyleObj.style;
// -->
</script>
</html>
```

ElementCSSInlineStyle.style

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

elementCSSInlineStyleObj.style

Description

The read-only property style of the ElementCSSInlineStyle object is of type CSSStyleDeclaration. style is a textual representation of the style for HTML elements.

Example

Listing 13.68 demonstrates reading the style property from the ElementCSSInlineStyle object.

Listing 13.68 Reading the style Property

```
<html>
<script language="JScript">
<!--
var cssStyleDef = elementCSSInlineStyleObj.style;
// -->
</script>
</html>
```

Rect

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The Rect object represents any Rect value. Rect reflects the values in the style properties. Modifications in the style properties are reflected in this object. Table 13.13 lists all properties of the Rect object.

Table 13.13 Properties Associated with the Rect Object

Item	Description
bottom	This read-only property is of type CSSPrimitiveValue. Represents the bottom of the Rect.
left	This read-only property is of type CSSPrimitiveValue. Represents the left of the Rect.

Item	Description
right	This read-only property is of type CSSPrimitiveValue. Represents the right of the Rect.
top	This is a read-only property of type CSSPrimitiveValue. Represents the top of the Rect.

Example

Listing 13.69 illustrates reading the properties of the Rect object.

Listing 13.69 *Reading Properties of the Rect Object*

```
<html>
<script language="JScript">
<!--
var rectObj = primitiveValObj.getRectValue();
var topVal = rectObj.top;
var rightVal = rectObj.right;
var bottomVal = rectObj.bottom;
var leftVal = rectObj.left;
// -->
</script>
</html>
```

Rect.bottom

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

rectObj.bottom

Description

The read-only property bottom of the Rect object is of type CSSPrimitiveValue. bottom represents the bottom of the Rect.

Example

Listing 13.70 illustrates reading the bottom property of the Rect object.

Listing 13.70 *Reading the bottom Property of the Rect Object*

```
<html>
<script language="JScript">
<!--
var rectObj = primitiveValObj.getRectValue();
var bottomVal = rectObj.bottom;
// -->
</script>
</html>
```

Rect.left

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Rect.left

Description

The read-only property left of the Rect object is of type CSSPrimitiveValue. Left represents the left of the Rect.

Example

Listing 13.71 illustrates reading the left property of the Rect object.

Listing 13.71 Reading the left Property of the Rect Object

```
<html>
<script language="JScript">
<!--
var rectObj = primitiveValObj.getRectValue();
var leftVal = rectObj.left;
// -->
</script>
</html>
```

Rect.right

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Rect.right

Description

The read-only property right of the Rect object is of type CSSPrimitiveValue. Right represents the right of the rectangle.

Example

Listing 13.72 illustrates reading the right property of the Rect object.

Listing 13.72 Reading the right Property of the Rect Object

```
<html>
<script language="JScript">
<!--
var rectObj = primitiveValObj.getRectValue();
var rightVal = rectObj.right;
```

```
// -->
</script>
</html>
```

Rect.top

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

Rect.top

Description

The read-only property top of the Rect object is of type CSSPrimitiveValue. The top property represents the top of the Rect.

Example

Listing 13.73 illustrates reading the top property of the Rect object.

Listing 13.73 Reading the top Property of the Rect Object

```
<html>
<script language="JScript">
<!--
var rectObj = primitiveValObj.getRectValue();
var topVal = rectObj.top;
// -->
</script>
</html>
```

RGBColor

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The RGBColor object is a representation of any RGB color value. RGBColor reflects the values in the style properties. Modifications in the style properties are reflected in this object. Table 13.14 lists all properties of the RGBColor object.

Table 13.14 *Properties Associated with the RGBColor Object*

Item	Description
blue	This read-only property is of type CSSPrimitiveValue. Represents the blue value of the RGB color.
green	This read-only property is of type CSSPrimitiveValue. Represents the green value of the RGB color.
red	This is a read-only property of type CSSPrimitiveValue. Represents the red value of the RGB color.

Example

Listing 13.74 illustrates the manipulation of the RGBColor object.

Listing 13.74 *Reading Properties of the RGBColor Object*

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColor();
var redVal = rgbColorObj.red;
var greenVal = rgbColorObj.green;
var blueVal = rgbColorObj.blue;

// -->
</script>
</html>
```

RGBColor.blue

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

```
rgbColorObj.blue
```

Description

The read-only property blue of the RGBColor object is of type CSSPrimitiveValue. blue is used to represent the blue value of the RGB color.

Example

Listing 13.75 illustrates the manipulation of the RGBColor object.

Listing 13.75 *Reading the blue Property of the RGBColor Object*

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColor();
```

```
var blueVal = rgbColorObj.blue;
// -->
</script>
</html>
```

RGBColor.green

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`rgbColor.green`

Description

The read-only property `green` of the `RGBColor` object is of type `CSSPrimitiveValue`. `green` is used to represent the green value of the RGB color.

Example

Listing 13.76 illustrates the manipulation of the `RGBColor` object.

Listing 13.76 Reading the green Property of the RGBColor Object

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColor();
var greenVal = rgbColorObj.green;
// -->
</script>
</html>
```

RGBColor.red

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

`rgbColor.red`

Description

The read-only property `red` of the `RGBColor` object is of type `CSSPrimitiveValue`. `Red` is used to represent the red value of the RGB color.

Example

Listing 13.77 illustrates the manipulation of the `RGBColor` object.

Listing 13.77 *Reading the red Property of the RGBColor Object*

```
<html>
<script language="JScript">
<!--
var rgbColorObj = primitiveValObj.getRGBColor();
var redVal = rgbColorObj.red;

// -->
</script>
</html>
```

ViewCSS

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

DOM CSS Object.

Description

The ViewCSS object represents a CSS view. The method associated with the ViewCSS object is as follows:

<u>Item</u>	<u>Description</u>
getComputedStyle	This method returns a CSSStyleDeclaration representing the computed style.

Example

Listing 13.78 demonstrates getting the computed style of a particular element.

Listing 13.78 *Computing the Style*

```
<html>
<script language="JScript">
<!--
viewCSSObj.getComputerStyle(bodyObj, null);
// -->
</script>
</html>
```

ViewCSS.getComputedStyle()

JavaScript 1.5+, JScript 5+
Nav6+, IE5+

Syntax

viewCSSObj.getComputedStyle(elt, pseudoElt)

Description

The `getComputedStyle()` method of the `ViewCSS` object takes the arguments *Element elt* and *String pseudoElt* and returns the computed style.

Example

Listing 13.79 demonstrates getting the computed style of a particular element.

Listing 13.79 Computing the Style Using the `getComputedStyle()` Method of the `ViewCSS` Object

```
<html>
<script language="JScript">
<!--
viewCSSObj.getComputerStyle(tableObj, null);
// -->
</script>
```




CHAPTER 14

JScript RunTime

This chapter contains all the items and elements making up Microsoft's Scripting Engine additions. This contains JScript elements that are specific to Microsoft's interpreters. These elements can be used with IIS ASP pages, Windows Script Host, and within the Internet Explorer browser. Table 14.1 shows the JScript support in the various applications that are shipped with Microsoft's Scripting Engine.

NOTE

Microsoft products provide the ability to upgrade the scripting engines, which are separate from the application, in its products. This means that it is possible for Internet Explorer 3 to have JScript 5.0 with full backward compatibility. Table 14.1 only lists the engine version that was shipped with the product.

Table 14.1 JScript Support in Various Microsoft Applications

Application	JScript Version
Internet Explorer 3.x	1.0
IIS 1.0	2.0
Internet Explorer 4.x	3.0
IIS 4.0	3.0
Windows Script Host 1.0	3.0
Visual Studio 6.0	4.0
Internet Explorer 5.0	5.0
IIS 5.0	5.1
Windows 2000	5.1
Internet Explorer 5.5	5.5

All the extensions discussed in this chapter are objects supplied in Microsoft's Scripting Runtime Library, which is named `scrrun.dll`. They are not intrinsic members of the JScript language itself, which is found in the `jscript.dll` library. However, there are some exceptions. These are the conditional compilation flags, `GetObject` and `ActiveXObject`, and the various `ScriptEngine` functions that are covered in Chapter 7, "Core Language."

The exceptions noted previously are probably the most notable differences between JScript and JavaScript. JScript is an automation-compatible language—it can create and manipulate *COM (Component Object Model)* objects. Both the `Dictionary` and `FileSystemObject` objects are examples of this, as are Excel, Word, and other Microsoft Office applications. The binaries for these applications, in the world of COM, are called *COM servers*.

It is worth noting that the Microsoft Scripting Runtime Library did not come with Internet Explorer until version 5. It was available with IIS and the Windows Script Host. As mentioned in the previous note, you can download the latest version from Microsoft's Scripting Technologies page at <http://msdn.microsoft.com/scripting>.

Now that you have taken a quick look at Microsoft's JScript and COM, you can get on to the reference. As with the other chapters in this section of the book, each entry includes a version, syntax, description, example, as well as many other details of the JScript language.

Dictionary

JScript3.0+

Syntax

```
dictionaryObj = new ActiveXObject("Scripting.Dictionary");
```

Description

The `Dictionary` object is an associative array of items that can be of any type. Each item is associated with a unique key that provides access to each item in the array. The key is usually an integer or a string, but can be anything except an array. A `Dictionary` object is created and returned when the `ActiveXObject()` method has the server and type name set to "Scripting.Dictionary".

A number of properties and methods are provided by the `Dictionary` object to provide access to the items in the dictionary. These properties and methods are listed in Table 14.2.

Table 14.2 *Properties and Methods Associated with Dictionary Object*

Type	Item	Description
Properties	Count	Returns the number of items in a collection or dictionary

Type	Item	Description
Methods	Item	Sets or returns an item for a specified key
	Key	Sets a key in a dictionary
	Add()	Adds a key and item pair to dictionary
	Exists()	Determines if specified key exists in dictionary
	Items()	Returns array of all items in dictionary
	Keys()	Returns array of all existing keys in dictionary
	Remove()	Removes a <i>key</i> , <i>item</i> pair from dictionary
	RemoveAll()	Removes all <i>key</i> , <i>item</i> pairs from dictionary

Example

Listing 14.1 creates a dictionary object using the `ActiveXObject()` constructor and assigns the new dictionary object to the variable "fruits".

Listing 14.1 Create a Dictionary Object Using `ActiveXObject()`

```
<script type="text/jscript" language="JScript">
<!--
var fruits = new ActiveXObject("Scripting.Dictionary");
// -->
</script>
```

Dictionary.Add()

JScript3.0+

Syntax

dictionaryobj.Add(*key*, *item*)

Description

The `Add()` method of the `Dictionary` object adds new items to the dictionary using *key*, *item* pairs. The *item* can be of any type, but the *key* cannot be an array. Nothing is returned from this method.

Example

Listing 14.2 creates a fruit dictionary and then uses the `Add()` method to add fruit items to the dictionary.

Listing 14.2 Create and Add Items to a Fruit Dictionary Using `Add()`

```
<script type="text/jscript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");
```

Listing 14.2 Continued

```
//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("O","orange");
// -->
</script>
```

Dictionary.Count

JScript3.0+

Syntax

dictionaryobj.Count

Description

The Count property of the Dictionary object contains the number of items in the dictionary. This property is read-only, so it cannot be used to change the size of the dictionary.

Example

Listing 14.3 creates a dictionary and displays the number of items that it contains.

Listing 14.3 Display the Number of Items in Dictionary Using Count

```
<script type="text/javascript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

document.write("There are ",fruits.Count," items in this dictionary.");
// -->
</script>
```

Dictionary.Exists()

JScript3.0+

Syntax

dictionaryobj.Exists(key)

Description

The `Exists()` method of the `Dictionary` object determines whether an item exists in the dictionary based on the *key* that is passed in to the method. If *key* exists, `true` is returned; otherwise `false` is returned from the method.

Example

Listing 14.4 creates a dictionary and determines if any fruits that begin with “s” are in the dictionary

Listing 14.4 Determine if “s” Fruits Exist in Dictionary Using `Exists()`

```
<script type="text/jscript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

if(fruits.Exists("S"))
document.write("The fruit ",fruits.Item('S')," exists in dictionary.");
else
document.write("No fruits that begin with <b>s</b> exist in dictionary.");
// -->
</script>
```

Dictionary.Item()

JScript3.0+

Syntax

```
dictionaryobj.Item(key)
dictionaryobj.Item(key) = newItem
```

Description

The `Item()` property of the `Dictionary` object provides the capability to retrieve, create, and modify items in the dictionary.

To retrieve an item from the dictionary, simply specify the *key* of the item to be returned in the argument list. The item associated with the *key* will be returned. If the *key* does not exist in the dictionary, a new *key*, *item* pair will be created in the dictionary and the item will be left empty. Nothing is returned from the method when this scenario occurs.

To create a new *key*, *item* pair in the dictionary, specify a new unique *key* as the method's argument. Directly after the `Item()` method, place an equal sign followed by the *newItem* to be associated with the *key*. The newly created item will be returned from the method.

Finally, an item can be changed by specifying the *key* associated with the item in the argument list. Directly after the `Item()` method, place an equal sign followed by the *newItem* that will replace the existing item. The new item will be returned from the method.

Example

Listing 14.5 uses `Item()` to create, modify, and retrieve an item from the dictionary.

Listing 14.5 Retrieving, Creating, and Modifying Items Using Item()

```
<script type="text/jscript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("O","orange");

//Use Item() method to create a new item in dictionary.
fruits.Item("G") = "grape";

//Use Item() method to change item associated with G.
fruits.Item("G") = "grapefruit"

//Return item associated with G and display it on screen.
document.write("The item ",fruits.Item('G')," is associated with G.");

// -->
</script>
```

Dictionary.Items()

JScript3.0+

Syntax

```
dictionaryobj.Items()
```

Description

The `Items()` method of the `Dictionary` object returns all the items in the dictionary in an Visual Basic array.

NOTE

A Visual Basic array (VBAArray object) can be converted to a JScript array by using the VBAArray object's toArray() method.

Example

Listing 14.6 uses the Items() method to retrieve all the items that in the dictionary as a Visual Basic array. The toArray() method is then used to convert the Visual Basic array to a JavaScript array. Each item in the new theArray array is displayed in the browser.

Listing 14.6 Retrieve Array of Items Using the Items() Method

```
<script type="text/jscript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

//Create an array.
theArray = (new VBAArray(fruits.Items())).toArray();

document.write("The array contains:<br>");
//Display items in array.
for (i in theArray)
{
document.write("theArray[" + i + "]=",theArray[i], "<br>");
}

// -->
</script>
```

Dictionary.Key()

JScript3.0+

Syntax

dictionaryobj.Key(key) + = newKey

Description

The `Key()` property of the Dictionary object provides the ability to change an existing key in the dictionary. To change a key, simply specify the *key* of the item you want changed followed by an equal sign and the *newKey*. If the *key* does not exist in the dictionary, a new *key*, *item* pair will be created in the dictionary using *newKey* and leaving the item empty. In either case, the new *key* is returned by the method.

Example

Listing 14.7 uses `Key()` to change an existing key in the dictionary.

Listing 14.7 *Changing Existing Key Using* `Key()`

```
<script type="text/javascript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A", "apple");
fruits.Add("B", "berry");
fruits.Add("G", "grape");
fruits.Add("O", "orange");

//Change key "O" to "R".
fruits.Key("O") = "R";

// -->
</script>
```

Dictionary.Keys()

JScript3.0+

Syntax

dictionaryobj.Keys()

Description

The `Keys()` method of the Dictionary object returns all the keys in the dictionary in an array.

Example

Listing 14.8 uses the `Keys()` method to retrieve all the keys that are in the dictionary.

Listing 14.8 *Retrieve Array of Keys Using the* `Keys()` *Method*

```
<script type="text/javascript" language="JScript">
<!--
```

```
//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

//Create an array.
theArray = (new VBAArray(fruits.Keys())).toArray();

document.write("The array contains:<br>");
//Display keys in array.
for (i in theArray)
{
document.write("theArray[" + i + "]=",theArray[i], "<br>");
}

// -->
</script>
```

Dictionary.Remove()

JScript3.0+

Syntax

dictionaryobj.Remove(key)

Description

The Remove() method of the Dictionary object removes the *key*, *item* pair from the dictionary that matches the *key* passed into the method. If no match is found, an error is returned. Nothing is returned from this method.

Example

Listing 14.9 uses the Remove() method to remove berry from the dictionary.

Listing 14.9 Remove the berry Item and Associated Key from the Dictionary Using Remove()

```
<script type="text/javascript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
```

Listing 14.9 Continued

```
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

//Delete berry from dictionary.
fruits.Remove("B");

// -->
</script>
```

Dictionary.RemoveAll()

JScript3.0+

Syntax

```
dictionaryobj.RemoveAll()
```

Description

The RemoveAll() method of the Dictionary object removes all *key, item* pairs from the dictionary. Nothing is returned from this method.

Example

Listing 14.10 uses RemoveAll() to remove items from the dictionary.

Listing 14.10 Remove All Items from the Dictionary Using RemoveAll()

```
<script type="text/jscript" language="JScript">
<!--

//Create dictionary.
var fruits = new ActiveXObject("Scripting.Dictionary");

//Define elements of dictionary.
fruits.Add("A","apple");
fruits.Add("B","berry");
fruits.Add("G","grape");
fruits.Add("O","orange");

//Delete all items from the dictionary.
fruits.RemoveAll();

// -->
</script>
```

Drive

JScript3.0+

Syntax

The core JScript object is created by calling the `GetDrive()` method of the `FileSystemObject` object.

Description

File System Objects (FSO) provide access to properties, methods, and events, using simple dot notation, to allow you to work with folders and files. A `Drive` is a file system object that provides access to the properties of disk drives and shared networks. A `Drive` object is created using the `GetDrive()` method of the `FileSystemObject` object. The properties associated with the `Drive` object are listed in Table 14.3. For more information on file system objects, see the `FileSystemObject` section in this chapter.

Table 14.3 Properties Associated with the `Drive` Object

Item	Description
<code>AvailableSpace</code>	Returns the amount of space available to a user on the specified drive or shared network
<code>DriveLetter</code>	Returns the drive letter of a physical local drive or a shared network
<code>DriveType</code>	Returns the value indicating the type of the specified drive
<code>FileSystem</code>	Returns the type of file system in use for the specified drive
<code>FreeSpace</code>	Returns the amount of free space available to a user on the specified drive or shared network
<code>IsReady</code>	Returns the status of the specified drive
<code>Path</code>	Returns the path for a specified file, folder, or drive
<code>RootFolder</code>	Returns a <code>Folder</code> object representing the root folder of a specified drive
<code>SerialNumber</code>	Returns the decimal serial number used to uniquely identify a disk volume
<code>ShareName</code>	Returns the shared network's name for a specified drive
<code>TotalSize</code>	Returns the total space of a drive or shared network
<code>VolumeName</code>	Sets or returns the volume name of the specified drive

Example

Listing 14.11 creates a `Drive` object using the `GetDrive()` method associated with the `FileSystemObject` object.

Listing 14.11 Create a `Drive` Object

```
<script type="text/jscript" language="Jscript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
```

Listing 14.11 Continued

```
//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));
// -->
</script>
```

Drive.AvailableSpace

JScript3.0+

Syntax

driveobject.AvailableSpace

Description

The AvailableSpace property of the Drive object contains the amount of space available to the user on the specified drive or shared network.

Example

Listing 14.12 displays the available space on drive C.

Listing 14.12 Find Available Space on Drive C Using the AvailableSpace Property of the Drive Object

```
<script type="text/javascript" language="JScript">
<!--

// Create a string to hold the drive path.
var drivePath = "C: ";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the available space.
document.write("Available space for drive ",drivePath," is ");
document.write(drive.AvailableSpace);

// -->
</script>
```

Drive.DriveLetter

JScript3.0+

Syntax

driveobject.DriveLetter

Description

The DriveLetter property of the Drive object contains the drive letter of the local drive or a shared network.

Example

Listing 14.13 retrieves the drive letter for the specified drive.

Listing 14.13 *Display the Drive Letter Using the DriveLetter Property of the Drive Object*

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:.";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the drive letter.
document.write("The driver letter for ",drivePath," is ",drive.DriveLetter);

// -->
</script>
```

Drive.DriveType

JScript3.0+

Syntax

driveobject.DriveType

Description

The DriveType property of the Drive object contains a numeric value indicating the type of the specified drive. The values associated with the DriveType are listed in Table 14.4.

Table 14.4 *Drive Type Value Definitions*

Value	Description
0	Unknown
1	Removable
2	Fixed
3	Network

Table 14.4 Continued

Value	Description
4	CD-ROM
5	RAM Disk

Example

Listing 14.14 retrieves the drive type for the specified drive.

Listing 14.14 Display the Drive Type Using the DriveType Property of the Drive Object

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the drive type.
document.write("The drive type for ",drivePath," is ",drive.DriveType);

// -->
</script>
```

Drive.FileSystem

JScript3.0+

Syntax

```
driveobject.FileSystem
```

Description

The FileSystem property of the Drive object contains the type of file system used by the specified drive

Example

Listing 14.15 retrieves the file system of the specified drive.

Listing 14.15 Display the File System Using the FileSystem Property of the Drive Object

```
<script type="text/jscript" language="JScript">
<!--
```

```
//Create a string to hold the drive path.
var drivePath = "C:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the file system used by the drive.
document.write("Drive ",drivePath," uses the ");
document.write(drive.FileSystem," file system.");

// -->
</script>
```

Drive.FreeSpace

JScript3.0+

Syntax

driveobject.FreeSpace

Description

The FreeSpace property of the Drive object contains the amount of free space, in bytes, available to the user on the specified drive or shared network.

Example

Listing 14.16 retrieves the free space associated with the C drive

Listing 14.16 Display the Free Space Using the FreeSpace Property of the Drive Object

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display free space on the drive.
document.write("Drive ",drivePath," has ",drive.FreeSpace);
document.write(" bytes of free space.");
```

Listing 14.16 Continued

```
// -->
</script>
```

Drive.IsReady

JScript3.0+

Syntax

driveobject.IsReady

Description

The IsReady property of the Drive object contains the status of the specified drive. If the drive is ready, true is returned; otherwise false is returned

Example

Listing 14.17 determines if drive A is ready.

Listing 14.17 Determine If Drive A Is Ready Using the IsReady Property of the Drive Object

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "A:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
//Create a Drive object for drive A.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Determine if drive is ready.
if(drive.IsReady)
document.write("Drive ",drivePath," is ready.");
else
document.write("Drive ",drivePath," is NOT ready.");
// -->
</script>
```

Drive.Path

JScript3.0+

Syntax

driveobject.Path

Description

The Path property of the Drive object contains the path of the specified file, folder, or drive.

Example

Listing 14.18 displays drive C's path.

Listing 14.18 *Display the Path of Drive C Using the Path Property of the Drive Object*

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the path of drive C.
document.write("The path of drive C is ",drive.Path);

// -->
</script>
```

Drive.RootFolder

JScript3.0+

Syntax

driveobject.RootFolder

Description

The RootFolder property of the Drive object contains a Folder object that represents the root folder of the specified drive.

Example

Listing 14.19 provides a function that returns the root folder of drive C.

Listing 14.19 *Return the Root Folder Using the RootFolder Property of the Drive Object*

```
function getCRootFolder()
{
    //Create a string to hold the drive path.
    var drivePath = "C:";
```

Listing 14.19 Continued

```
//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Return root folder object for drive C.
return(drive.RootFolder);
}
```

Drive.SerialNumber

JScript3.0+

Syntax

driveobject.SerialNumber

Description

The SerialNumber property of the Drive object contains the decimal serial number that uniquely identifies the specified disk volume.

Example

Listing 14.20 displays the unique serial number of drive C.

Listing 14.20 Displaying the Unique Serial Number of Drive C Using the SerialNumber Property of the Drive Object

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";
//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display the serial number associated with the drive.
document.write("Drive serial number is ",drive.SerialNumber);

// -->
</script>
```

Drive.ShareName

JScript3.0+

Syntax

`driveobject.ShareName`

Description

The `ShareName` property of the `Drive` object contains the network share name for the specified drive.

Example

Listing 14.21 provides a function that returns the share name of a drive.

Listing 14.21 Return the Share Name Using the `ShareName` Property of the `Drive` Object

```
function getShareName(drivePath)
{
    //Create a file system object.
    var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    //Create a Drive object for the drivePath.
    var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

    //Return share name of the drive.
    return(drive.ShareName);
}
```

Drive.TotalSize

JScript3.0+

Syntax

`driveobject.TotalSize`

Description

The `TotalSize` property of the `Drive` object returns the total size of the specified drive in bytes.

Example

Listing 14.22 shows the total size of the drive.

Listing 14.22 Return the Total Size of Drive Using the `TotalSize` Property of the `Drive` Object

```
<script type="text/jscript" language="JScript">
<!--

//Create a string to hold the drive path.
var drivePath = "C:";
```

Listing 14.22 Continued

```
//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Create a Drive object for drive C.
var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

//Display total space on the drive.
document.write("Drive ",drivePath," is ",drive.TotalSize," bytes.");

// -->
</script>
```

Drive.VolumeName

JScript3.0+

Syntax

```
driveobject. VolumeName
driveobject.VolumeName = newVolumeName
```

Description

The VolumeName property of the Drive object contains the volume name of the specified drive. If an equal sign follows this property, the volume name is set to *newVolumeName*.

Example

Listing 14.23 provides a function that returns the volume name of a drive.

Listing 14.23 Return the Volume Name Using the VolumeName Property of the Drive Object

```
function getVolumeName(drivePath)
{
    //Create a file system object.
    var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    //Create a Drive object for the drivePath.
    var drive = fileSysObj.GetDrive(fileSysObj.GetDriveName(drivePath));

    //Return volume name of the drive.
    return(drive.VolumeName);
}
```

Drives

JScript3.0+

Syntax

fileSysObj.Drives

Description

The `Drives` collection holds a read-only collection of all the available drives. A collection is similar to an array except that an item pointer is used to navigate through the items in the collection rather than an array index. You can only move the current item pointer to the first or next element of a collection. Because the `Drive` object is a collection, the drives can only be accessed by using an `Enumerator` object. The properties associated with the `Drives` object are listed in Table 14.5. For more information on accessing collections, see the `Enumerator` object.

Table 14.5 *Properties Associated with the `Drives` Collection*

Item	Description
Count	Returns the number of items in the collection
Item	Returns an item for a specified key in a <code>Drives</code> Collection object

Example

Listing 14.24 accesses the number of drives in the `Drives` collection using the `FileSystemObject` and the `Drives` object `count` property.

Listing 14.24 Access a `Drives` Object

```
<script type="text/jscript" language="JScript">
<!--
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
document.write("There are ",fileSysObj.Drives.count," drives");
// -->
</script>
```

Drives.Count

JScript3.0+

Syntax:

`Drives.Count`

Description:

The `Count` property of the `Drives` collection contains the number of items in the collection.

Example

Listing 14.25 displays the number of items contained in the `Drives` collection.

Listing 14.25 *Display the Number of Items in the Drives Collection Using Its Count Property*

```
<script type="text/jscript" language="JScript">
<!--

//Create a file system object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");

//Display number of drives.
document.write("There are ",fileSysObj.Drives.count," drives.");

// -->
</script>
```

Drives.Item()

JScript3.0+

Syntax

```
new enumeratorobj = Enumerator(Drives)
enumeratorobj.Item(key)
```

Description

The Item() property returns a drive, specified by *key*, from the enumerated Drives collection.

Example

Listing 14.26 begins by creating an Enumerator object to access the Drives collection. The Item() property is used by the enumerator object to access the letter and type of each drive.

Listing 14.26 *Retrieving Drives Using Item()*

```
<html>
<h1>Drive Letters and drive types:</h1>

<script type="text/jscript" language="JScript">
<!--
//Create enumerator object.
var fileSysObj = new ActiveXObject("Scripting.FileSystemObject");
var en = new Enumerator(fileSysObj.Drives);

//Display drive letter and type for each drive.
for (;!en.atEnd();en.moveNext())
{
    document.write("Drive ",en.item().DriveLetter);
    document.write(" is of type ",en.item().DriveType,"<br>");
}
}
```

```
// -->  
</script>  
</html>
```

File

JScript3.0+

Syntax

```
fileObj = new ActiveXObject("Scripting.FileSystemObject");
```

Description

The File object provides access to all the properties of a file. The File object has specific properties and methods associated with it, as shown in Table 14.6.

Table 14.6 *Properties and Methods of the File Object*

Property/Method	Description
Attributes	Refers to the attributes of the file
Copy()	Copies a file from one location to another
Created	Returns the date the file was created
DateLastAccessed	Returns the date the file was last accessed
DateLastModified	Returns the date the file was last modified
Delete()	Removes a file
Drive	Returns the drive on which the file exists
Move()	Moves a file from one location to another
Name	Returns the name for a file
OpenAsTextStream()	Opens a text stream for a file
ParentFolder	Returns the parent folder name for the file
Path	Returns the path to the file
ShortName	Returns the short name of the file
ShortPath	Returns the short path of the file
Size	Returns the size of the file
Type	Returns the file type

Example

Listing 14.27 shows how to create a new File object.

Listing 14.27 *Example of the File Object*

```
<script type="text/jscript" language="JScript">  
!..  
  
var myObject, f;
```

Listing 14.27 Continued

```
// Create instance of new FileSystemObject.  
myObject = new ActiveXObject("Scripting.FileSystemObject");  
  
f = myObject.GetFile("f:\\test.txt");  
document.write("The name of the file is: " + f.Name);  
// -->  
</script>
```

File.Attributes

JScript3.0+

Syntax

```
file.Attributes[newattributes]
```

Description

The Attributes property is used to determine or set the attributes of a file. This property is an array that takes the optional parameter *newattributes* to set any new attributes. Attributes can be read/write or read-only. The list of valid attributes is shown in Table 14.7.

Table 14.7 Attribute Properties

Property	Description
0	Specifies a Normal file. No attributes set.
1	Specifies that a file is read-only. Attribute is read/write.
2	Specifies that a file is hidden. Attribute is read/write.
4	Refers to a system file. Attribute is read/write.
8	Refers to the disk drive volume label. Attribute is read-only.
16	Refers to a folder in a directory. Attribute is read-only.
32	Specifies that a file has changed since the last backup. Attribute is read/write.
64	Refers to a link or shortcut. Attribute is read-only.
128	Refers to the disk drive volume label. Attribute is read-only.

Example

Listing 14.28 retrieves all the file attributes of a fictitious file called test.txt. When the user clicks the Get Attributes button, the Attributes property is used to retrieve the properties and display them in an alert box.

Listing 14.28 Example of Attributes

```
<html>  
  
<script type="text/jscript" language="JScript">
```

```
<!--
function get()
{
    // declare variables.
    var myObject, f;

    // Create a new File object.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");

    // Check for all the attributes of the file.
    if(!f.attributes) alert("no attributes set");
    var strAttributes = "";
    if (f.attributes & 1) strAttributes += "Read only\n";
    if (f.attributes & 2) strAttributes += "Hidden\n";
    if (f.attributes & 4) strAttributes += "System\n";
    if (f.attributes & 8) strAttributes += "Volume label\n";
    if (f.attributes & 16) strAttributes += "Folder\n";
    if (f.attributes & 32) strAttributes += "Archive bit set\n";
    if (f.attributes & 64) strAttributes += "Shortcut or link\n";
    if (f.attributes & 128) strAttributes += "File is compressed\n";
    alert(strAttributes);
}
// -->
</script>

Get a files attributes.<br>
<form name="myForm">
<input type="Button" value="Get Attributes" onClick='get()'>
</form>

</html>
```

File.Copy()

JScript3.0+

Syntax

file.Copy(destination, overwrite)

Description

The `Copy()` method is used to copy a specified file from one location to another. This method takes two parameters. *destination* is the location to which to copy the file. The *overwrite* argument is an optional Boolean value indicating whether to overwrite an existing file of the same name should one exist.

Example

Listing 14.29 uses the Copy() method to copy the fictitious test.txt file to "testcopy.txt".

Listing 14.29 *Example of Copy()*

```
<html>
<script type="text/jscript" language="JScript">
<!-- Hide

function copy()
{
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");

    // Copy the file "test.txt" to a file called "testcopy.txt"
    f.Copy("c:\\testcopy.txt",1);
}

// End Hide -->
</script>
```

Press the Copy button to copy the test.txt file to testcopy.txt

```
<form name="myForm">
<input type="Button" value="Copy" onClick='copy() '>
</form>
</html>
```

File.DateCreated

JScript3.0+

Syntax

file.DateCreated

Description

The DateCreated property is used to get the date when the file was created.

Example

Listing 14.30 shows how to use to DateCreated property to get the creation date of the test.txt file.

Listing 14.30 *Example of DateCreated*

```
<html>
<script type="text/jscript" language="JScript">
<!--
```

```
// Function gets the date of when the file "mytest.txt" was
// created.
function get()
{
    var myObject, f, date;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");

    // Determine the date when the file was created.
    date = f.DateCreated;
    alert ("The date this file was created is: " + date);
}
//-->
</script>
```

Get the date that test.txt was created.


```
<form name="myForm">
<input type="Button" value="Get Date" onClick='get()'>
</form>

</html>
```

File.DateLastAccessed

JScript3.0+

Syntax

file.DateLastAccessed

Description

The DateLastAccessed property is used to find out the last date when the file was accessed.

Example

Listing 14.31 shows how the DateLastAccessed property is used to retrieve the date when the test.txt file was last accessed.

Listing 14.31 Example of DateLastAccessed

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function determines the date of when the file was
// last accessed.
```

Listing 14.31 Continued

```
function get()
{
    var myObject, f, date;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");
    date = f.DateLastAccessed;

    // Inform the user of the date when the file was last accessed.
    alert ("The date this file was last accessed is: " + date);
}
//-->
</script>
```

Get the date that test.txt was last accessed.


```
<form name="myForm">
<input type="Button" value="Get Date" onClick='get()'>
</form>
</html>
```

File.DateLastModified

JScript3.0++

Syntax

file.DateLastModified

Description

The DateLastModified property returns the date when the file was last modified.

Example

Listing 14.32 shows how the DateLastModified property is used to retrieve the date when the test.txt file was last modified.

Listing 14.32 Example of DateLastModified

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function creates a new file object and the determines when
// it was last modified.
function get()
{
    var myObject, f, date;
```



```
// Create an instance of the FileSystemObject.
myObject = new ActiveXObject("Scripting.FileSystemObject");
f = myObject.GetFile("c:\\test.txt");
date = f.DateLastModified;

// Inform the user of the date when the file was last modified.
alert ("The date this file was last modified is: " + date);
}
//-->
</script>
```

Get the date that test.txt was last modified.


```
<form name="myForm">
<input type="Button" value="Get Date" onClick='get()'>
</form>

</html>
```

File.Delete

JScript3.0+

Syntax

```
file.Delete()
```

Description

The Delete() method is used to delete a specified file.

Example

Listing 14.33 shows how to delete a file using the Delete() method.

Listing 14.33 Example of the Delete() Method

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function creates a new file and then deletes it using the
// delete method.
function remove()
{
    var myObject;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    var f = myObject.GetFile("c:\\mytest.txt");
    f.Delete();
}
```

Listing 14.33 Continued

```
}  
//-->  
</script>
```

Delete the file "mytest.txt".


```
<form name="myForm">  
<input type="Button" value="Delete File" onClick='remove()'>  
</form></html>
```

File.Drive

JScript3.0++

Syntax

file.Drive

Description

The Drive property returns the drive letter of the drive on which the specified file or folder resides.

Example

Listing 14.34 shows how the Drive property is used to get the letter of the drive that contains the specified file.

Listing 14.34 Example of the Drive Property

```
<html>
```

```
<script type="text/jscript" language="JScript">  
!--
```

```
// Function gets the drive letter on which the file resides and  
// then informs the user of the drive.
```

```
function get()
```

```
{  
    var myObject, f, d;  
    // Create an instance of the FileSystemObject.  
    myObject = new ActiveXObject("Scripting.FileSystemObject");  
    f = myObject.GetFile("c:\\test.txt");  
    d = f.drive;  
    alert("The drive that test.txt resides is: " + d);  
}
```

```
//-->  
</script>
```

Get Drive for test.txt


```
<form name="myForm">
<input type="Button" value="Get Drive" onClick='get()''>
</form>

</html>
```

File.Move()

JScript3.0+

Syntax

file.Move(*destination*)

Description

The Move method is used to move a file from one location to another. This method takes a single parameter, *destination*, which represents the location to which the file is to be moved.

Example

Listing 14.35 uses the Move() method to move the test.txt file from the "C:\\\" directory to the "C:\\mydir\\" directory.

Listing 14.35 *Example of the Move Method*

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function creates a new file and then moves it from the
// C drive to the E drive.
function move()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");
    f.Move("c:\\mydir\\");
}
//-->
</script>
```

Press the button to move the file test.txt into the mydir directory.


```
<form name="myForm">
<input type="Button" value="Move File" onClick='move()''>
</form>

</html>
```

File.Name

JScript3.0++

Syntax

file.Name

Description

The Name property is used to either set or retrieve the name of a file object.

Example

Listing 14.36 uses the Name property to retrieve the filename (test.txt) and display it in an alert box.

Listing 14.36 *Example of the Name Property*

```
<html>

<script type="text/jscript" language="JScript">
<!--
// Function creates a new file and uses the name property
// to display the name of the newly created file.
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\test.txt");
    alert("The name of the file is: " + f.Name);
}
//-->
</script>
Get the name of the file object.<br>
<form name="myForm">
<input type="Button" value="Get Name" onClick='get() '>
</form>

</html>
```

File.OpenAsTextStream()

JScript3.0+

Syntax

file.OpenAsTextStream(*iomode*, *format*)

Description

The OpenAsTextStream method is used to open a specified text stream that can be used to read or write to a file. This method takes two optional parameters. The first parameter, *iomode*, specifies the mode for the file. Valid modes are ForReading, ForWriting,

and ForAppending as shown in Table 14.8. The second parameter, *format*, specifies the format of the file. The default value is ASCII: However you can have one of the three settings shown in Table 14.9.

Table 14.8 *Settings for the iomode Parameter*

Constant	Value	Description
ForReading	1	Opens the file for reading only.
ForWriting	2	Opens the file for reading and writing. If the file already exists then its contents are overwritten.
ForAppending	8	Opens the file for reading and writing. If the file already exists, new content is added to the end of the file.

Table 14.9 *Settings for the format Parameter*

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII mode.

Example

Listing 14.37 uses the OpenAsTextStream method to open a stream to the file test.txt in ASCII mode and then write a string to it.

Listing 14.37 *Example of OpenAsTextStream*

```
<html>

<script type="text/jscript" language="JScript">
<!--

function open()
{
    var myObject, f, text;

    // Define the variables locally.
    var TristateFalse = 0;
    var ForWriting = 2;

    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    myObject.CreateTextFile( "c:\\test.txt" );
    f = myObject.GetFile("c:\\test.txt");
    text = f.OpenAsTextStream(ForWriting, TristateFalse);
    text.Write( "Just some nonsense" );
    text.Close( );
}
//-->
```

Listing 14.37 Continued

```
</script>
Open a file and write some text to it.<br>

<form name="myForm">
<input type="Button" value="open a filestream" onClick='open() '>
</form>

</html>
```

File.ParentFolder

JScript3.0++

Syntax

`file.ParentFolder`

Description

The ParentFolder property is used to determine the parent folder from which the file was obtained.

Example

Listing 14.38 shows how the ParentFolder property is used.

Listing 14.38 Example of ParentFolder

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function displays the path to the name of the parent folder.
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The name of the parent folder is: " + f.ParentFolder.Path);
}
//-->
</script>
```

Click to get the name of the parent folder.


```
<form name="myForm">
<input type="Button" value="Get Parent Folder" onClick='get() '>
</form>

</html>
```

File.Path

JScript3.0++

Syntax

file.Path

Description

The Path property is used to get the path for a specified file.

Example

Listing 14.39 shows an example of the Path property.

Listing 14.39 Example of the Path Property

```
<html>

<script type="text/jscript" language="JScript">
<!--
function getPath()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The name of the path is: " + f.Path);
}
//-->
</script>
Get the path to the file.<br>

<form name="myForm">
<input type="Button" value="Get Path" onClick='getPath()'>
</form>

</html>
```

File.ShortName

JScript3.0++

Syntax

file.ShortName

Description

The ShortName property returns the short name used by programs that require the earlier 8.3 naming convention.

Example

Listing 14.40 shows how to access the short name of the file.

Listing 14.40 *Example of the ShortName Property*

```
<html>

<script type="text/jscript" language="JScript">
<!--

// Function gets the shortname for the file.
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The file ShortName is: " + f.ShortName);
}
//-->
</script>
Get the file's short name.<br>

<form name="myForm">
<input type="Button" value="Get Shortname" onClick='get()'/>
</form>

</html>
```

File.ShortPath

JScript3.0++

Syntax

file.ShortPath

Description

The ShortPath property returns the short path used by programs that require the earlier 8.3 naming convention.

Example

Listing 14.41 shows how to get the short path for the specified file.

Listing 14.41 *Example of ShortPath*

```
<html>

<script type="text/jscript" language="JScript">
<!--
```



```
// Function gets the shortpath of the file object.
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The file ShortPath name is: " + f.ShortPath);
}
//-->
</script>
Get the files short path.<br>

<form name="myForm">
<input type="Button" value="Get Shortpath" onClick='get()'>
</form>

</html>
```

File.Size

JScript3.0++

Syntax

file.Size

Description

The Size property returns the size of the specified file.

Example

In Listing 14.42, the Size property is used to get the file size.

Listing 14.42 *Example of Size*

```
<html>

<script type="text/jscript" language="JScript">
<!--
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The file size is: " + f.Size);
}
//-->
```

Listing 14.42 Continued

```
</script>
Get the size of the file.<br>
<form name="myForm">
<input type="Button" value="Get File Size" onClick='get()''>
</form>

</html>
```

File.Type

JScript3.0++

Syntax

file.Type

Description

The Type property is used to get information pertaining to the type of file.

Example

Listing 14.43 shows an example of how the Type property is used to return the type of the myTest.txt file.

Listing 14.43 Example of the Type Property

```
<html>

<script type="text/jscript" language="JScript">
<!--
function get()
{
    var myObject, f;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Gets the file myTest.txt.
    f = myObject.GetFile("c:\\tmp\\myTest.txt");
    alert("The name type is: " + f.Type);
}
//-->
</script>

<form name="myForm">
<input type="Button" value="Get File Type" onClick='get()''>
</form>

</html>
```

Files

JScript3.0+

Syntax

Core JScript Collection of File objects.

Description

The Files object represents a collection of files in a folder. The Files object has specific properties and methods associated with it, which are shown in Table 14.10.

Table 14.10 *Properties and Methods of the Files Object*

Property/Method	Description
Count	Returns the number of items in a collection
Item	Sets or returns an item based on a specific key

Example

Listing 14.44 shows how to create a new Files object.

Listing 14.44 *Example of Files Object*

```
<html>

<script language="JScript">
<!--

// Function displays a list of the files.
// in the specified folder.
function ShowList(foldername)
{
    var myObject, f, MyFiles, names;
// Create an instance of the FileSystemObject.
myObject = new ActiveXObject("Scripting.FileSystemObject");
f = myObject.GetFolder(foldername);

// Stores the files in an enumerator.
myFiles = new Enumerator(f.files);
names="";
for (i=0; !myFiles.atEnd(); myFiles.moveNext()){
    names += myFiles.item();
    names += "<br>";
}
document.write(names);
}

// -->
</script>
```

Listing 14.44 Continued

```
<body>
<form name="form1">
Enter the name of an existing folder
including the drive letter if it is not on the C drive.
<input type="text" size="35" name="file">
<br><br>
<input type="Button" value="Get File List"
onClick='ShowList(document.form1.file.value)'\>
</form>

</body>
to </html>
```

Files.Count

JScript2.0+

Syntax

files.Count

Description

The Count property returns the number of items in the Files collection.

Example

Listing 14.45 shows how to use the Count property to get the number of items in the Files collection.

Listing 14.45 Example of the Count Property

```
<html>

<script language="JScript">
<!--

// Function gets the count of the file collection.
// in a specified folder.
function getCount(foldername)
{
    var myObject, f, filesCount;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    f = myObject.GetFolder(foldername);

    // Gets the number of files in a folder.
    filesCount = f.files.Count;
```

```
    document.write("The number of files in this folder is: " + filesCount);
}

// -->
</script>

<body>
<form name="myForm">

Enter the name of a folder which contains files.
<input type="text" size="35" name="file">
<br><br>
<input type="Button" value="Get File Count"
onClick='getCount(document.form1.file.value)'\>
</form>

</body>
</html>
```

Files.Item

JScript2.0+

Syntax

files.Item

Description

The Item property sets or returns an *item* based on a specified *key* in a collection of files.

Example

Listing 14.46 shows how to use the Item property.

Listing 14.46 Example of the Item Property

```
<html>

<script language="JScript">
<!--

// Function gets the Item of the file collection.
// in a specified folder.
function getItem(foldername)
{
    var myObject, f, filesItem;

    // Create instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
```

Listing 14.46 Continued

```
f = myObject.GetFolder(foldername);

// Gets the setup.exe file.
filesItem = f.files.Item("Setup.exe ");

document.write("The file that you got is: " + filesItem);
}
// -->
</script>

<body>
<form name="myForm">
Enter the name of a folder which contains files.
<input type="text" size="35" name="file">
<br><br>
<input type="Button" value="Get File Item"
onClick='getItem(document.form1.file.value) '>
</form>

</body>
</html>
```

FileSystemObject

JScript3.0+

Syntax

Core JScript object created by calling the ActiveXObject() method passing Scripting.FileSystemObject.

Description

The FileSystemObject object provides access to the computer's file system. The FileSystemObject object has specific properties and methods associated with it, which are shown in Table 14.11.

Table 14.11 Properties and Methods of the FileSystemObject Object

Property/Method	Description
BuildPath()	Appends information to a file path
CopyFile()	Copies a file from one location to another
CopyFolder()	Copies a folder from one location to another
CreateFolder()	Creates a new folder object
CreateTextfile()	Creates a new text file object
DeleteFile()	Removes a file
DeleteFolder()	Removes a folder object

Property/Method	Description
DriveExists()	Determines whether a drive exists
Drives	Returns a <code>Drives</code> collection, containing all the available drive objects
FileExists()	Determines whether a file exists
FolderExists()	Determines whether a folder exists
GetAbsolutePathName()	Returns the absolute pathname for a file
GetBaseName()	Gets the base name of the last component
GetDrive()	Gets the drive letter for a file
GetDriveName()	Gets the drive name on which a file resides
ExtensionName()	Returns the extension for a file
GetFile()	Gets the file object
GetFileName()	Gets the name of a file
GetFolder()	Gets the folder name that contains a file
GetParentFolderName()	Gets the parent folder's name
GetSpecialFolder()	Gets the folder names for special folders
GetTempName()	Creates a randomly generated temporary file
MoveFile()	Moves a file from one location to another
MoveFolder()	Moves a folder and its contents from one location to another
OpenTextFile()	Opens a text file stream to a file

Example

Listing 14.47 shows how to create a new `FileSystemObject` object.

Listing 14.47 *Example of a `FileSystemObject` Object*

```
<html>

<script language="JScript">
<!--

// create a new FileSystemObject
var myObject;
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// -->
</script>

</html>
```

FileSystemObject.BuildPath()

JScript3.0+

Syntax

filesystemobject.BuildPath(*path*, *name*)

path—String representing existing path

name—Name being appended to existing path

Description

The BuildPath() method is used to append a name to an existing path. A separator is inserted between the existing path and the appended name if necessary.

Example

Listing 14.48 shows how to use the BuildPath() method.

Listing 14.48 *Example of* BuildPath()

```
<html>

<script language="JScript">
<!--

function GetNewPath(path)
{
    var myObject, newpath;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    newpath = myObject.BuildPath(path, "\\newstuff");
    alert("The result is: " + newpath);
}
// -->
</script>
```

Append "\\newstuff" to the existing path: "c:\\tmp"

```
<body>
<form name="myForm">
<input type="Button" value="Build Path" onClick='GetNewPath("c:\\tmp")'>
</form>

</body>
</html>
```

FileSystemObject.CopyFile

JScript3.0+

Syntax

filesystemobject.CopyFile(*source*, *destination*, *overwrite*)

Description

The `CopyFile()` method is used to copy one or more files to a specified directory. This method takes three parameters. The first parameter, *source*, is a string specifying the source path and filename from which to copy. The second parameter, *destination*, is a string specifying the destination path and filename to which to copy. The final parameter, *overwrite*, is an optional parameter. It represents a Boolean value indicating whether to overwrite an existing file.

Example

Listing 14.49 shows how to copy a file from one location to another.

Listing 14.49 *Example of* `CopyFile()`

```
<html>

<script language="JScript">
<!--

function copy()
{
    var myObject, newpath;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Copies file test.txt from the root C drive to the tmp folder.
    myObject.CopyFile ("c:\\test.txt", "c:\\tmp\\myTest.txt");
}

// -->
</script>

<body>
Copy "c:\\test.txt" to the existing path: "c:\\tmp\\myTest.txt"

<form name="myForm">
<input type="Button" value="Copy File" onClick='copy()'>
</form>

</body>
</html>
```

FileSystemObject.CopyFolder()

JScript3.0+

Syntax

`filesystemobject.CopyFolder(source, destination, overwrite)`

Description

The `CopyFolder()` method copies a folder from a source to a destination. This method takes three parameters. The first parameter, *source*, is a string specifying the source path and filename from which to copy. Wildcards can be used to copy more than one folder. The second parameter, *destination*, is a string specifying the destination path and filename to which to copy. The final parameter, *overwrite*, is an optional parameter. It represents a Boolean value indicating whether to overwrite an existing file.

Example

Listing 14.50 shows how to use the `CopyFolder()` method. A new `FileSystemObject` is created, providing access to the computer's file system. It is then used to copy all the contents of the folder `tmp` to `newtmp`. The `*` specifies that all the contents of `tmp` are to be copied.

Listing 14.50 *Example of* `CopyFolder()`

```
<html>

<script language="JScript">
<!--

// Function creates a new FileSystemObject and copies
// the file from one folder to another.
function copy()
{
    var myObject;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Copies all files and folders from "tmp" to "newtmp".
    myObject.CopyFolder ("c:\\tmp\\*", "c:\\newtmp\\");
}

// -->
</script>
<body>
Copy Folder "c:\\tmp" to the folder "c:\\newtmp"

<form name="myForm">
<input type="Button" value="Copy Folder" onClick='copy()'/>
</form>

</body>
</html>
```

FileSystemObject.CreateFolder()

JScript3.0+

Syntax

filesystemobject.CreateFolder(*name*)

Description

The CreateFolder() method is used to create a new folder. This method takes the *name* parameter, which is a string specifying the folder name to create.

Example

Listing 14.51 shows how to use the CreateFolder() method.

Listing 14.51 Example of CreateFolder()

```
<html>

<script language="JScript">
<!--

// Function creates a new folder called "newtmp".
function create()
{
    var myObject, newfolder;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// Creates the folder newtmp.
    newfolder = myObject.CreateFolder ("c:\\newtmp\\");
}

// -->
</script>
<body>
Create Folder: "c:\\newtmp"

<form name="myForm">
<input type="Button" value="Click to Create New Folder" onClick='create()''>
</form>

</body>
</html>
```

FileSystemObject.CreateTextFile()

JScript2.0+

Syntax

`filesystemobject.CreateTextFile(filename, overwrite, unicode)`

Description

The `CreateTextFile()` method is used to create a new text file. This method takes two parameters. The first parameter, *filename*, is a string specifying the name of the file to create, whereas the second parameter, *overwrite*, is a Boolean value indicating whether to overwrite an existing file. A third parameter, *unicode*, is an optional parameter that indicates whether the file should be created as Unicode or ASCII. It is a Boolean value with a default value of false. To create a file as Unicode, set this value to true.

Example

Listing 14.52 shows how to use the `CreateTextFile()` method. An instance of the `FileSystemObject` is created, which in turn uses the `CreateTextFile()` method to create the file `testing.txt`.

Listing 14.52 *Example of* `CreateTextFile()`

```
<html>

<script language="JScript">
<!--

// Function creates a new text file called "testing.txt".
function createFile()
{
    var myObject, newfile;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// Creates a text file called "testing.txt".
    newfile = myObject.CreateTextFile("c:\\testing.txt", false);
}

// -->
</script>
<body>
Create Text file "testing.txt"

<form name="myForm">
<input type="Button" value="Click to Create New File" onClick='createFile()'>
</form>

</body>
</html>
```

FileSystemObject.DeleteFile()

JScript3.0+

Syntax

```
filesystemobject.DeleteFile(filename, force)
```

Description

The DeleteFile() method is used to delete an existing file. An instance of the FileSystemObject is created, which uses the DeleteFile() method to remove the file testing.txt. This method takes two parameters. The first parameter, *filename*, is a string specifying the name of the file to delete, whereas *force* is a Boolean value indicating whether a file that has read-only permissions can be deleted.

Example

Listing 14.53 shows how to delete a file.

Listing 14.53 Example of DeleteFile()

```
<html>

<script language="JScript">
<!--

// Function deletes the file "testing.txt".
function deleteFile()
{
    var myObject;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// Delete the file testing.txt.
    myObject.DeleteFile("c:\\testing.txt");
}

// -->
</script>
<body>
Delete Text file "testing.txt"

<form name="myForm">
<input type="Button" value="Click to Delete File" onClick='deleteFile()''>
</form>

</body>
</html>
```

FileSystemObject.DeleteFolder()

JScript3.0+

Syntax

```
filesystemobject.DeleteFolder(foldername, force)
```

Description

The DeleteFolder() method is used to remove an existing folder and all its contents. This method takes two parameters. The first parameter, *foldername*, is a string specifying the name of the folder to delete, whereas *force* is a Boolean value indicating whether a file that has read-only permissions can be deleted.

Example

Listing 14.54 shows how the DeleteFolder() method is used. An instance of the FileSystemObject is created, which uses the DeleteFolder() method to remove the folder newtmp.

Listing 14.54 Example of DeleteFolder()

```
<html>

<script language="JScript">
<!--

// Function deletes the folder "newtmp".
function deleteFolder()
{
    var myObject;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// Delete the folder "newtmp".
    myObject.DeleteFolder("c:\\newtmp");
}

// -->
</script>
<body>
Delete Folder: "newtmp"

<form name="myForm">
<input type="Button" value="Click to Delete Folder" onClick='deleteFolder()''>
</form>

</body>
</html>
```

FileSystemObject.DriveExists()

JScript3.0+

Syntax

filesystemobject.DriveExists (*letter*)

Description

The DriveExists() method is used to determine whether a drive exists. This method takes *letter* as a parameter, which specifies a drive letter to look for.

Example

Listing 14.55 shows how to check for the existence of a D drive.

Listing 14.55 Example of DriveExists()

```
<html>

<script language="JScript">
<!--

// Function checks for the existence of drive D.
function checkDrive()
{
    var myObject;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// Check for the existence of a D drive.
    if(myObject.DriveExists("d")){
        alert("Drive D exists");
    }
    else {
        alert("Drive D doesn't exist");
    }
}

// -->
</script>
<body>
Check for drive "D"

<form name="myForm">
<input type="Button" value="Check drive" onClick='checkDrive()''>
</form>

</body>
</html>
```

FileSystemObject.Drives

JScript3.0+

Syntax

filesystemobject.Drives

Description

The Drives collection holds all the Drive objects. It is used to get an object consisting of all the drives.

Example

Listing 14.56 shows how to check for drives.

Listing 14.56 *Example of* Drives

```
<html>

<script language="JScript">
<!--

function getdrive()
{
    var myObject, drive, e;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    drive = myObject.Drives;

    // Store drives in an enumerator e.
    e = new Enumerator(drive);
    alert("The first drive is: " + e.item());
}

// -->
</script>
<body>
<form name="myForm">
<input type="Button" value="Get Drive Name" onClick='getdrive()'/>
</form>

</body>
</html>
```

FileSystemObject.FileExists()

JScript3.0+

Syntax

```
filesystemobject.FileExists(filename)
```

Description

The `FileExists()` method is used to determine whether a file exists on the current system. This method takes *filename* as its only parameter, which represents the name of the file to check.

Example

Listing 14.57 shows how to use the `FileExists()` method to check to see whether the file `test.txt` exists.

Listing 14.57 Example of `FileExists()`

```
<html>

<script language="JScript">
<!--

function checkfile()
{
    var myObject;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Check to see if the file exists and alert user of status.
    if(myObject.FileExists("c:\\test.txt")){
        alert("File Exists");
    }
    else {
        alert("File doesn't exist");
    }
}

// -->
</script>
<body>
    Check for file "test.txt"

    <form name="myForm">
    <input type="Button" value="Check file" onClick='checkfile() '>
    </form>

</body>
</html>
```

FileSystemObject.FolderExists()

JScript3.0+

Syntax:

filesystemobject.FolderExists(foldername)

Description

The `FolderExists()` method is used to determine whether a specified folder exists on the current system. This method takes *foldername* as its only parameter, which represents the name of folder to check.

Example

Listing 14.58 shows an example of how to determine whether the tmp folder exists on a specific drive.

Listing 14.58 *Example of* `FolderExists()`

```
<html>

<script language="JScript">
<!--

function checkfolder()
{
    var myObject;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Check for the tmp folder.
    if(myObject.FolderExists("c:\\tmp")){
        alert("tmp Folder Exists on the C drive");
    }
    else {
        alert("tmp Folder doesn't exist on the C drive");
    }
}

// -->
</script>
<body>
Check for folder "tmp" on the C drive

<form name="myForm">
<input type="Button" value="Check Folder" onClick='checkfolder()''>
</form>
```

```
</body>
</html>
```

FileSystemObject.GetAbsolutePathName()

JScript3.0+

Syntax

```
filesystemobject.GetAbsolutePathName()
```

Description

The `GetAbsolutePathName()` method returns a complete and unambiguous path from a provided path specification.

Example

Listing 14.59 shows how to use the `GetAbsolutePathName()` method.

Listing 14.59 *Example of* `GetAbsolutePathName()`

```
<html>

<script language="JScript">
<!--

// Function gets the absolute path and informs the user.
function getpath()
{
    var myObject, path;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    path = myObject.GetAbsolutePathName("c:\\");
    alert("Absolute Path Name is: " + path);
}

// -->
</script>
<body>
Get the Absolute Path for C:\

<form name="myForm">
<input type="Button" value="Get Absolute Path" onClick='getpath()'/>
</form>

</body>
</html>
```

FileSystemObject.GetBaseName()

JScript3.0+

Syntax

filesystemobject.GetBaseName(*path*)

path—Required parameter specifying the component whose base name is to be returned.

Description

The GetBaseName() method returns a string containing the base name of a file or folder, without any extension, in a path.

Example

Listing 14.60 shows how the GetBaseName() method is used.

Listing 14.60 *Example of* GetBaseName()

```
<html>

<script language="JScript">
<!--

// Function gets the Base name of the FileSystemObject.
function get()
{
    var myObject, b;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Gets the specified base name.
    b = myObject.GetBaseName("c:\\tmp\\myTest.txt");
    alert("The base name is: " + b);
}

// -->
</script>
<body>
<form name="myForm">
<input type="Button" value="Get Base" onClick='get() '>
</form>

</body>
</html>
```

FileSystemObject.GetDrive()

JScript3.0+

Syntax

```
filesystemobject.GetDrive(drive)
```

Description

The GetDrive() method is used to determine the drive specified in the drive parameter. This method takes *drive* as its only parameter, which represents the letter of drive for which to get information.

Example

Listing 14.61 shows how the GetDrive() method is used.

Listing 14.61 Example of GetDrive()

```
<html>

<script language="JScript">
<!--

function getdrive()
{
    var myObject, drive;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    drive = myObject.GetDrive("c:\\");
    alert("Drive Name of drive C is: " + drive.VolumeName);
}

// -->
</script>
<body>
Get the name for Drive C.

<form name="myForm">
<input type="Button" value="Get Drive Name" onClick='getdrive()'/>
</form>

</body>
</html>
```

FileSystemObject.GetDriveName()

JScript3.0+

Syntax

`filesystemobject.GetDriveName(letter)`

Description

The `GetDriveName()` method is used to get the name of a specified drive. This method takes *drive* as its only parameter, which represents the letter of drive for which to get the name.

Example

Listing 14.62 shows how the `GetDriveName()` method is used. An instance of the `FileSystemObject` is created and then used to get the name of the C drive.

Listing 14.62 *Example of* `GetDriveName()`

```
<html>

<script language="JScript">
<!--

function getdrive()
{
    var myObject, drive;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Gets the name of the specified drive.
    drive = myObject.GetDriveName("c:\\");

    // Informs the user of the drive name.
    alert("Name for drive C is: " + drive);
}

// -->
</script>
<body>
Get the name for Drive C.

<form name="myForm">
<input type="Button" value="Get Drive Name" onClick='getdrive()'>
</form>

</body>
</html>
```

FileSystemObject.GetExtensionName()

JScript3.0+

Syntax

filesystemobject.GetExtensionName(*path*)

Description

The GetExtensionName() method is used to get the file extension name for a specified file. This method takes *path* as its only parameter, which represents the full path to the file.

Example

Listing 14.63 shows how the GetExtensionName() method is used. An instance of the FileSystemObject is created and then used to get the extension name of the specified file type.

Listing 14.63 *Example of* GetExtensionName()

```
<html>

<script language="JScript">
<!--
// Gets the extension name for the file and informs the user
// of the results.
function getExt()
{
    var myObject, ext;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    //Gets the extension of the specified file.
    ext = myObject.GetExtensionName("c:\\testing.txt");

    //Informs the user of the file extension.
    alert("The extension for the testing file is: " + ext);
}

// -->
</script>
<body>
Get the extension for the file "testing.txt".

<form name="myForm">
<input type="Button" value="Get Extension" onClick='getExt()'/>
</form>

</body>
</html>
```

FileSystemObject.GetFile()

JScript3.0+

Syntax

filesystemobject.GetFile()

Description

The GetFile() method is used to get a specified file object.

Example

Listing 14.64 shows how you can use the GetFile() method to get the size of a file.

Listing 14.64 *Example of* GetFile()

```
<html>

<script language="JScript">
<!--

function getsize()
{
    var myObject, afile, size;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    afile = myObject.GetFile("c:\\test.txt")

    //Gets the size of the file.
    size = afile.Size;

    // Informs user of the file size.
    alert("The size of the test.txt file is:" + size + " Bytes.");
}

// -->
</script>
<body>
Get the size for the file "test.txt"

<form name="myForm">
<input type="Button" value="Get Size" onClick='getsize()'/>
</form>

</body>
</html>
```

FileSystemObject.GetFileName()

JScript3.0+

Syntax

filesystemobject.GetFileName(filename)

Description

The `GetFileName()` method is used to get a specified filename. This method takes *filename* as its only parameter, which represents the absolute or relative path to the file.

Example

Listing 14.65 shows how the `GetFileName()` property is used.

Listing 14.65 *Example of* `GetFileName()`

```
<html>

<script language="JScript">
<!--

function getname()
{
    var myObject, name;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Gets the file name.
    name = myObject.GetFileName("c:\\test.txt");

    // Informs the user of the name.
    alert("The file name is:" + name);
}

// -->
</script>
<body>
Get the name value for the file "test.txt"

<form name="myForm">
<input type="Button" value="Get File Name" onClick='getname()'/>
</form>

</body>
</html>
```

FileSystemObject.GetFolder()

JScript3.0+

Syntax

`filesystemobject.GetFolder()`

Description

The `GetFolder()` method is used to get the specified folder object.

Example

Listing 14.66 shows how to retrieve a folder object. An instance of the `FileSystemObject` is created and then used to get a folder name. Then the `DateLastAccessed` property is applied to the folder object to get the actual date that the folder was last accessed. The result is then returned to the user.

Listing 14.66 *Example of* `GetFolder()`

```
<html>

<script language="JScript">
<!--

// Function gets the folder object and then checks the date.
// Property that identifies when the folder was last accessed.
function getfolder()
{
    var myObject, afolder, date;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Gets the folder object.
    afolder = myObject.GetFolder("c:\\tmp");

    // Gets the date the folder was last accessed.
    date = afolder.DateLastAccessed;

    // Informs the user of the date result.
    alert("The date the folder" + afolder + " was last accessed is:" + date);
}

// -->
</script>
<body>
Get the date that the folder "tmp" was last accessed.

<form name="myForm">
<input type="Button" value="Get Folder Date" onClick='getfolder() '>
</form>

</body>
</html>
```

FileSystemObject.GetParentFolderName()

JScript3.0+

Syntax

```
filesystemobject.GetParentFolderName(foldername)
```

Description

The GetParentFolderName() method is used to get the name of the parent folder for a specified file or folder. This method takes *foldername* as its only parameter, which represents the path for the name of the subfolder.

Example

Listing 14.67 shows how to get the name for the parent folder.

Listing 14.67 Example of GetParentFolderName()

```
<html>

<script language="JScript">
<!--

function getparent()
{
    var myObject, name;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    name = myObject.GetParentFolderName("c:\\tmp\\subfolder");
    alert("The parent folder name is:" + name);
}

// -->
</script>
<body>
Get the parent folder name for "subfolder" folder.

<form name="myForm">
<input type="Button" value="Get Parent Folder" onClick='getparent()'/>
</form>

</body>
</html>
```

FileSystemObject.GetSpecialFolder()

JScript3.0+

Syntax

`filesystemobject.GetSpecialFolder(num)`

Description

The `GetSpecialFolder()` method is used to get the special folder object specified. This method takes *num* as its only parameter, which represents one of the following values:

- 0—Windows folder
- 1—System folder
- 2—Temporary folder

Example

Listing 14.68 shows how the `GetSpecialFolder()` method is used to get the Temporary folder.

Listing 14.68 *Example of* `GetSpecialFolder()`

```
<html>

<script language="JScript">
<!--

function get()
{
    var myObject, name;
    // Creates an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");
    name = myObject.GetSpecialFolder(2);
    alert("The folder is a temporary folder" + name);
}

// -->
</script>
<body>
Check to see if the folder "Temp" is a temporary folder.

<form name="myForm">
<input type="Button" value="Get Special Folder" onClick='get()'>
</form>

</body>
</html>
```

FileSystemObject.GetTempName()

JScript3.0+

Syntax

`filesystemobject.GetTempName()`

Description

The `GetTempName()` method is used to get a randomly generated temporary file or folder.

Example

Listing 14.69 shows how the `GetTempName()` method is used. An instance of the `FileSystemObject` is created and then used to get the name of a randomly generated temporary file.

Listing 14.69 *Example of* `GetTempName()`

```
<html>

<script language="JScript">
<!--

function get()
{
    var myObject, name;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Get the name of a temporary file.
    name = myObject.GetTempName();
    alert("The temp file name is: " + name);
}

// -->
</script>
<body>
Get a temp file.

<form name="myForm">
<input type="Button" value="Get Temp Name" onClick='get() '>
</form>

</body>
</html>
```

FileSystemObject.MoveFile()

JScript3.0+

Syntax

filesystemobject.MoveFile(*source*, *destination*)

Description

The MoveFile() method is used to move a file from a source to a destination. This method takes two parameters. The first parameter, *source*, is the location of the file to be moved, whereas the second parameter, *destination*, is the new location of the moved file.

Example

Listing 14.70 shows how the MoveFile() method is used. An instance of the FileSystemObject is created and then used to move test.txt from the root directory on the C drive to the tmp folder.

Listing 14.70 *Example of* MoveFile()

```
<html>

<script language="JScript">
<!--

function move()
{
    var myObject;
// Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

// File is moved from root to tmp folder.
    myObject.MoveFile("c:\\test.txt", "c:\\tmp\\test.txt");
}

// -->
</script>
<body>
Move the file "test.txt" from c:\ to c:\tmp

<form name="myForm">
<input type="Button" value="Move file" onClick='move()'/>
</form>

</body>
</html>
```

FileSystemObject.MoveFolder()

JScript3.0+

Syntax

`filesystemobject.MoveFolder()`

Description

The `MoveFolder()` method is used to move one or more folders from one location to another.

Example

Listing 14.71 shows how the `MoveFolder()` method is used. An instance of the `FileSystemObject` is created and then used to move the `subtmp` folder to the root directory on the C drive.

Listing 14.71 Example of `MoveFolder()`

```
<html>

<script language="JScript">
<!--

function move()
{
    var myObject;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Folder is moved.
    myObject.MoveFolder("c:\\tmp\\subtmp", "c:\\");
}

// -->
</script>
<body>
Move the folder "subtmp" from c:\tmp\subtmp to c:\

<form name="myForm">
<input type="Button" value="Move folder" onClick='move()'/>
</form>

</body>
</html>
```

FileSystemObject.OpenTextFile()

JScript2.0+

Syntax

`filesystemobject.OpenTextFile(filename, iomode, create, format)`

Description

The `OpenTextFile()` method is used to open a text stream object to a specified file, which can be used to read and write from that file. This method takes four parameters. The first parameter, *filename*, is the name of the file to be created. The second parameter, *iomode*, is an optional parameter indicating the mode of the file. The third parameter, *create*, is an optional Boolean value parameter indicating whether the file can be created if it doesn't exist. And the final parameter, *format*, is an optional parameter indicating the format of the opened file. The default is ASCII.

Example

Listing 14.72 shows how the `OpenTextFile()` method is used. An instance of the `FileSystemObject` is created and then used to open a text stream to the file `dummy.txt`. The file is opened in append mode.

Listing 14.72 *Example of* `OpenTextFile()`

```
<html>

<script language="JScript">
<!--

function open()
{
    var myObject, afile;
    // Create an instance of the FileSystemObject.
    myObject = new ActiveXObject("Scripting.FileSystemObject");

    // Create a File called "dummy.txt" and open for appending.
    afile = myObject.OpenTextFile("c:\\dummy.txt", ForAppending, false);
    afile.close();
}

// -->
</script>
<body>
Open a text stream for the file dummy.txt

<form name="myForm">
<input type="Button" value="Open File" onClick='open()'>
</form>

</body>
</html>
```


Folder

JScript3.0+

Syntax

Core JScript object created by calling the `GetFolder()` method of the `FileSystemObject` object.

Description

The `Folder` object, when created, provides access to all the properties of a folder. This object is created by using the `GetFolder()` method of the `FileSystemObject` object. The *folder* passed is the name the folder for which you want to create a JScript object.

Table 14.12 contains a list of methods and properties of the `Folder` object.

Table 14.12 Methods and Properties of the `Folder` Object

Type	Item	Description
Method	<code>Copy()</code>	Copies the folder from one location to another
	<code>Delete()</code>	Deletes the specified folder
	<code>Move()</code>	Moves the specified folder
Property	<code>Attributes</code>	Sets or returns the file system attributes of the folder
	<code>DateCreated</code>	Returns the date the folder was created
	<code>DateLastAccessed</code>	Returns the date the folder was last accessed
	<code>DateLastModified</code>	Returns the date the folder was last modified
	<code>Drive</code>	Returns the drive on which the folder is located
	<code>Files</code>	Returns a <code>Files</code> collection containing all the <code>File</code> objects in the folder
	<code>IsRootFolder</code>	Returns <code>true</code> if the folder is the root folder, <code>false</code> otherwise
	<code>Name</code>	Sets or returns the name of the folder
	<code>ParentFolder</code>	Returns a folder object for the parent folder of the folder
	<code>Path</code>	Returns the path to the folder
	<code>ShortName</code>	Returns the 8.3 version of the name of the folder
	<code>ShortPath</code>	Returns the 8.3 version of the path to the folder
	<code>Size</code>	Returns the byte size of all the contents of the folder
	<code>SubFolders</code>	Returns a <code>Folders</code> collection of all the folders contained in the folder
	<code>Type</code>	Returns the type of folder

Example

Listing 14.73 creates an instance of the `FileSystemObject` object, and then creates an instance of the `Folder` object. The drive letter of the folder instance is written to the user's page if this script is run in a browser window.

Listing 14.73 *Creating an Instance of the Folder Object*

```
<script type="text/jscript" language="JScript">

    // Create an instance of the FileSystemObject.
    var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    // Create an instance of the Folder object for the "temp" folder.
    var myFolder = myFileSysObj.GetFolder("\\temp");

    // Write the drive the folder appears to a web page.
    document.write(myFolder.Drive);

</script>
```

Folder.Attributes

JScript3.0+

Syntax

folder.Attributes

folder.Attributes = *attrib*

Description

The `Attributes` property of an instance of the `Folder` object is used to set or retrieve the attributes of a folder. If the property is called, as in the first syntactical definition, you can retrieve the current attributes of the folder. If you set the property equal to `attrib`, you are able to set new attributes. Table 14.13 has the values of the attributes.

Table 14.13 *Attribute Values*

Value	Stands For	Description
0	Normal	Normal folder with no attributes set
1	Read Only	Read-only folder with read/write attribute
2	Hidden	Hidden folder with read/write attribute
4	System	System folder with read/write attribute
8	Volume	Disk drive volume label with read-only attribute
16	Directory	Folder or directory with read-only attribute
32	Archive	Folder has changed since last backup with read/write attribute

Value	Stands For	Description
64	Alias	Link or shortcut to a folder with read-only attribute
128	Compressed	Compressed folder with read-only attribute

Example

In Listing 14.74 when the button is clicked, an alert box pops up displaying the attribute setting of the folder specified in the script. See the entry for `File.Attributes` for a more detailed example.

Listing 14.74 Using the `Attributes` Property

```
<html>
<head>
  <script type="text/jscript" language="JScript">

    function get(){

      // Create an instance of the FileSystemObject.
      var myObject = new ActiveXObject("Scripting.FileSystemObject");

      // Create an instance of a Folder object.
      var myFolder = myObject.GetFolder("c:\\Temp");

      // Display an alert box with the folder attributes.
      alert(myFolder.Attributes);
    }

  </script>
</head>
<body>

  Gets a folder's attributes.

  <form name="myForm">
    <input type="Button" value="Get Attributes" onClick='get()'>
  </form>

</body>
</html>
```

Folder.Copy()

JScript3.0+

Syntax

`folder.Copy(destination)`

`folder.Copy(destination, boolean)`

Description

The `Copy()` method is used to copy a specified folder from one location to another. The *destination* specified is the location to which to copy the folder. If *boolean* is set to true and a folder exists in the *destination* location, the copy will overwrite the folder.

Example

Listing 14.75 creates an instance of the `Folder` object, and then makes a copy using the `Copy()` method. If the new folder already exists, it will be overwritten.

Listing 14.75 Using the `Copy()` Method

```
// Create a FileSystemObject.
var myObject = new ActiveXObject("Scripting.FileSystemObject");

// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Temp");

// Copy the folder to another location. Overwrite the folder
// if it already exists.
myFolder.Copy("c:\\Test", true);
```

Folder.DateCreated

JScript3.0+

Syntax

folder.DateCreated

Description

The `DateCreated` property of an instance of the `Folder` object is used to get the date when the folder was created.

Example

Listing 14.76 can be run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the creation date of the folder specified.

Listing 14.76 Using the `DateCreated` Property

```
<html>
<head>
  <script language="JScript" type="text/jscript">

    function get(){

      // Create a FileSystemObject object.
      var myObject = new ActiveXObject("Scripting.FileSystemObject");
```

```
// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Temp");

// Display the creation date of the folder.
alert (myFolder.DateCreated);
}

</script>
<head>
<body>

Get the date the c:\\temp folder was created.

<form name="myForm">
  <input type="Button" value="Get Date" onClick='get()'>
</form>

</body>
</html>
```

Folder.DateLastAccessed

JScript3.0+

Syntax

folder.DateLastAccessed

Description

The DateLastAccessed property of an instance of the Folder object is used to get the date the folder was last accessed.

Example

Listing 14.77 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the date the folder was last accessed.

Listing 14.77 Using the DateLastAccessed Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

  function get(){

    // Create a FileSystemObject object.
    var myObject = new ActiveXObject("Scripting.FileSystemObject");
```

Listing 14.77 Continued

```
// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Temp");

// Display the date the folder was last accessed.
alert (myFolder.DateLastAccessed);
}

</script>
</head>
<body>
Get the date the c:\\temp folder was last accessed.

<form name="myForm">
  <input type="Button" value="Get Last Access" onClick='get()'>
</form>

</body>
</html>
```

Folder.DateLastModified

JScript3.0+

Syntax

folder.DateLastModified

Description

The DateLastModified property of an instance of the Folder object is used to get the date the folder was last modified.

Example

Listing 14.78 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the date the folder was last modified.

Listing 14.78 Using the DateLastModified Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

function get(){

    // Create a FileSystemObject object.
    var myObject = new ActiveXObject("Scripting.FileSystemObject");
```

```
// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Temp");

// Display the date the folder was last modified.
alert (myFolder.DateLastModified);
}

</script>
</head>
<body>
Get the date the c:\\temp folder was last modified.

<form name="myForm">
  <input type="Button" value="Get Last Modified" onClick='get()'>
</form>

</body>
</html>
```

Folder.Delete()

JScript3.0+

Syntax

folder.Delete(*boolean*)

folder.Delete()

Description

The Delete() method of an instance of the Folder object is used to delete a specified folder. The optional *boolean* parameter the method takes will cause the method to delete all read-only files and folders in the folder if set to true.

Example

Listing 14.79 deletes the specified folder.

Listing 14.79 Using the Delete() Method

```
// Create a FileSystemObject object.
var myObject = new ActiveXObject("Scripting.FileSystemObject");

// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Test");

// Delete the folder.
myFolder.Delete();
```

Folder.Drive

JScript3.0+

Syntax

folder.Drive

Description

The Drive property of an instance of the Folder object returns the drive letter of the folder.

Example

Listing 14.80 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the drive on which the folder is located.

Listing 14.80 *Using the Drive Property*

```
<html>
  <head>

    <script language="JScript" type="text/jscript">

      function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Temp");

        // Display the drive of the folder.
        alert (myFolder.Drive);
      }

    </script>
  </head>
  <body>

    Get the drive the c:\\temp folder is located on.

    <form name="myForm">
      <input type="Button" value="Get Drive" onClick='get()'>
    </form>

  </body>
</html>
```


Folder.Files

JScript3.0+

Syntax

folder.Files

Description

The Files property of an instance of the Folder object returns a collection consisting of all the File objects contained in the *folder*. This includes hidden and system files.

Example

Listing 14.81 creates an instance of the Folder object, and then creates an instance of the Enumerator object to access the collection of files contained in the Files property. If this example is loaded in a browser, each of the files contained in the folder will be written to the page.

Listing 14.81 Using the Files Property

```
<script language="JScript" type="text/jscript">

    // Create an instance of the FileSystemObject.
    var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    // Create an instance of the Folder object for the "temp" folder.
    var myFolder = myFileSysObj.GetFolder("\\temp");

    // Use an instance of the Enumerator object to access the files
    // in this folder.
    var myEnum = new Enumerator(myFolder.Files);

    // Write file names of the files in this folder.
    while(!myEnum.atEnd()){
        document.write(myEnum.item() + '<br>');
        myEnum.moveNext();
    }

</script>
```

Folder.IsRootFolder

JScript3.0+

Syntax

folder.IsRootFolder

Description

The IsRootFolder property of an instance of the Folder object, evaluates to true if the folder is the root folder, and false otherwise.

Example

Listing 14.82 creates an instance of the Folder object, and then checks to see if it is the root folder. If this example is loaded in a browser, a string will be written to the user's page stating whether the folder is the root folder.

Listing 14.82 Using the *IsRootFolder* Property

```
<script language="JScript" type="text/jscript">

    // Create an instance of the FileSystemObject.
    var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    // Create an instance of the Folder object for the "temp" folder.
    var myFolder = myFileSysObj.GetFolder("\\temp");

    // See if the folder is the root folder.
    if(myFolder.IsRootFolder){
        document.write(myFolder.Name + " is the root folder.");
    }else{
        document.write(myFolder.Name + " is not the root folder.");
    }

</script>
```

Folder.Move()

JScript3.0+

Syntax

folder.Move(*destination*)

Description

The Move() method is used to move a specified folder from one location to another. The *destination* specified is the location to which to move the folder.

Example

Listing 14.83 creates an instance of the Folder object, and then moves it by using the Move() method.

Listing 14.83 Using the *Move()* Method

```
// Create a FileSystemObject.
var myObject = new ActiveXObject("Scripting.FileSystemObject");

// Create a Folder object.
var myFolder = myObject.GetFolder("c:\\Temp");
```

```
// Move the folder to another location.  
myFolder.Move("c:\\Test");
```

Folder.Name

JScript3.0+

Syntax

folder.Name

Description

The Name property of an instance of the Folder object is used to either set or get the name of the folder.

Example

Listing 14.84 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the name of the folder.

Listing 14.84 *Example of the* Name *Property*

```
<html>  
<head>  
  
  <script language="JScript" type="text/javascript">  
  
    function get(){  
  
      // Create a FileSystemObject object.  
      var myObject = new ActiveXObject("Scripting.FileSystemObject");  
  
      // Create a Folder object.  
      var myFolder = myObject.GetFolder("c:\\Temp");  
  
      // Display the name of the folder.  
      alert (myFolder.Name);  
    }  
  
  </script>  
</head>  
<body>  
  Get the name of the c:\temp folder.  
  
  <form name="myForm">  
    <input type="Button" value="Get Name" onClick='get()'>  
  </form>  
  
</body>  
</html>
```

Folder.ParentFolder

JScript3.0+

Syntax

folder.ParentFolder

Description

The ParentFolder property of an instance of the Folder object is used to return a new Folder object for the parent folder.

Example

Listing 14.85 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the name of the parent folder.

Listing 14.85 *Using the ParentFolder Property*

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Temp");

        // Display the name of the parent folder.
        alert (myFolder.ParentFolder.Path);
    }

</script>
</head>
<body>
Get the name of the c:\\temp parent folder.

<form name="myForm">
    <input type="Button" value="Get Parent Folder" onClick='get()'>
</form>

</body>
</html>
```

Folder.Path

JScript3.0+

Syntax

folder.Path

Description

The Path property of an instance of the Folder object returns the full path to the folder.

Example

Listing 14.86 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the path of the folder.

Listing 14.86 *Using the Path Property*

```
<html>
  <head>

    <script language="JScript" type="text/jscript">

      function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Temp");

        // Display the path of the folder.
        alert (myFolder.Path);
      }

    </script>
  </head>
  <body>
    Get the path of the c:\\temp folder.

    <form name="myForm">
      <input type="Button" value="Get Path" onClick='get()'>
    </form>

  </body>
</html>
```

Folder.ShortName

JScript3.0+

Syntax

folder.ShortName

Description

The ShortName property of an instance of the Folder object returns the “8.3” name (an eight-character name, followed by a dot, ., and completed with a three-character extension) of the folder.

Example

Listing 14.87 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the short name of the folder.

Listing 14.87 Using the ShortName Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Program Files");

        // Display the short name of the folder.
        alert (myFolder.ShortName);
    }

</script>
</head>
<body>

Get the short name of the "C:\\Program Files" folder.

<form name="myForm">
    <input type="Button" value="Get Short Name" onClick='get()'/>
</form>

</body>
</html>
```

Folder.ShortPath

JScript3.0+

Syntax

folder.ShortPath

Description

The ShortPath property of an instance of the Folder object returns the “8.3” path (an eight-character name, followed by a dot, ., and completed with a three-character extension) to the folder.

Example

Listing 14.88 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the short path to the folder.

Listing 14.88 Using the ShortPath Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Program Files");

        // Display the short path of the folder.
        alert (myFolder.ShortPath);
    }

</script>
</head>
<body>
Get the short path of the "c:\\Program Files" folder.

<form name="myForm">
    <input type="Button" value="Get Short Path" onClick='get()'/>
</form>

</body>
</html>
```

Folder.Size

JScript3.0+

Syntax

folder.Size

Description

The `Size` property of an instance of the `Folder` object returns the byte size of the folder.

Example

Listing 14.89 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the byte size of the folder.

Listing 14.89 Using the `Size` Property

```
<html>
  <head>

    <script language="JScript" type="text/jscript">

      function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Program Files");

        // Display the size of the folder.
        alert (myFolder.Size);
      }

    </script>
  </head>
  <body>
    Get the size of the "c:\\Program Files" folder.

    <form name="myForm">
      <input type="Button" value="Get Size" onClick='get()'>
    </form>

  </body>
</html>
```

Folder.SubFolders

JScript3.0+

Syntax

`folder.SubFolders`

Description

The `SubFolders` property of an instance of the `Folder` object returns a collection consisting of all the `Folder` objects contained in the *folder*. This includes hidden and system folders.

Example

Listing 14.90 creates an instance of the `Folder` object, and then creates an instance of the `Enumerator` object to access the collection of folders contained in the `Folders` property. If this example is loaded in a browser, the names of each of the folders contained in the folder will be written to the page.

Listing 14.90 Using the *SubFolders* Property

```
<script language="JScript" type="text/jscript">

    // Create an instance of the FileSystemObject.
    var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

    // Create an instance of the Folder object for the "temp" folder.
    var myFolder = myFileSysObj.GetFolder("\\temp");

    // Use an instance of the Enumerator object to access the sub folders
    // in this folder.
    var myEnum = new Enumerator(myFolder.SubFolders);

    // Write file names of the folders in this folder.
    while(!myEnum.atEnd()){
        document.write(myEnum.item() + '<br>');
        myEnum.moveNext();
    }

</script>
```

Folder.Type

JScript3.0+

Syntax

folder.Type

Description

The `Type` property of an instance of the `Folder` object returns the type of the folder. For a folder, the return value is `File Folder`.

Example

Listing 14.91 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the type of the folder.

Listing 14.91 Using the Type Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create a FileSystemObject object.
        var myObject = new ActiveXObject("Scripting.FileSystemObject");

        // Create a Folder object.
        var myFolder = myObject.GetFolder("c:\\Program Files");

        // Display the type of the folder.
        alert (myFolder.Type);
    }

</script>
</head>
<body>
Get the type of the "c:\\Program Files" folder.

<form name="myForm">
    <input type="Button" value="Get Type" onClick='get() '>
</form>

</body>
</html>
```

Folders

JScript3.0+

Syntax

Core JScript Collection of Folder objects.

Description

The Folders collection holds a read-only collection of all the available folders. A collection is similar to an array except that an item pointer is used to navigate through the items in the collection rather than an array index. As you would with arrays, you can only move the current item pointer to the first or next element of a collection. Because the Folder object is a collection, the folders can only be accessed by using an Enumerator object. The properties associated with the Folders collection are listed in Table 14.14. For more information on accessing collections, see the Enumerator section in this chapter.

Table 14.14 *Properties Associated with Folders Collection*

Item	Description
Add	Allows you to add a folder to the collection
Count	Returns the number of items in the collection
Item	Sets or returns an item for a specified key in a Drives dictionary object

Example

Listing 14.92 accesses the Folders collection by using FileSystemObject.

Listing 14.92 *Accessing the Number of Items in the Folders Collection*

```
<script language="JScript" type="text/jscript">
    var fileSysObj = new ActiveXObject("Scripting.FileSystemObject ");
    document.write(fileSysObj.Folders.count);
</script>
```

Folders.Add()

JScript3.0+

Syntax

```
folderscollection.Add(foldername)
```

Description

The Add() method of the Folders collection is used to add folders to the collection. This can be used to create new folders on the file system.

Example

Listing 14.93 creates a test folder inside the \temp folder on the file system. This script can be executed by using the Windows Script Host (wscript.exe or cscript.exe). Be sure that you are on a drive that contains a folder named temp, and make sure that there is no folder named test already present. Not doing so will result in an error.

Listing 14.93 *Using the Add() Method of the Folders Collection*

```
// Create an instance of the FileSystemObject.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create an instance of the Folder object for the "temp" folder.
var myFolder = myFileSysObj.GetFolder("\\temp");

// Access the sub folders collection of our folder.
var myFolderCollection = myFolder.SubFolders;

// Add a new folder called "test".
myFolderCollection.Add("test");
```

Folders.Count

JScript3.0+

Syntax

folderscollection.Count

Description

The Count property of the Folders collection returns the number of folders in the collection. Folders are added to this collection by using the Folders.Add() method.

Example

Listing 14.94 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the number of folders in the Folders collection.

Listing 14.94 Accessing the Count Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create an instance of the FileSystemObject.
        var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

        // Create two instances of the Folder object for the "temp" folder.
        var myFolder = myFileSysObj.GetFolder("c:\\Temp");

        // Access the sub folders collection of our folder.
        var myFolderCollection = myFolder.SubFolders;

        // Add two new folders.
        myFolderCollection.Add("test1");
        myFolderCollection.Add("test2");

        // Display the number of folders in the collection.
        alert(myFolderCollection.Count);

    }

</script>
</head>
<body>
Get the count of the c:\\temp Folders collection.
```

```
<form name="myForm">
  <input type="Button" value="Get Count" onClick='get()'>
</form>

</body>
</html>
```

Folders.Item

JScript3.0+

Syntax

folderscollection.Item(*foldername*)

Description

The Item property of an instance of the Folders collection returns *foldername* folder.

Example

Listing 14.95 is run in an Internet Explorer browser, and it contains a single button. When the button is clicked, an alert box pops up displaying the second folder added to the Folders collection.

Listing 14.95 Accessing the Item Property

```
<html>
<head>

<script language="JScript" type="text/jscript">

    function get(){

        // Create an instance of the FileSystemObject.
        var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

        // Create two instances of the Folder object for the "temp" folder.
        var myFolder = myFileSysObj.GetFolder("c:\\Temp");

        // Access the sub folders collection of our folder.
        var myFolderCollection = myFolder.SubFolders;

        // Add two new folders.
        myFolderCollection.Add("test1");
        myFolderCollection.Add("test2");

        // Display the second folder added to the collection.
        alert(myFolderCollection.Item(2).Name);

    }
```

Listing 14.95 Continued

```

    </script>
</head>
<body>
Get the second item added to the c:\temp Folders collection.

<form name="myForm">
    <input type="Button" value="Get Item" onClick='get()'>
</form>

</body>
</html>
```

TextStream

JScript2.0+

Syntax

Core JScript object created by calling the CreateTextFile() method of the FileSystemObject object.

Description

The TextStream object is created by calling the CreateTextFile() method of an instance of the FileSystemObject. An instance of the TextStream object is created by calling the CreateTextFile() method of the FileSystemObject object.

Table 14.15 contains a list of methods and properties of the TextStream object.

Table 14.15 Methods and Properties of the TextStream Object

Type	Item	Description
Method	Close()	Closes the file
	Read()	Reads in a specified number of characters
	ReadAll()	Reads the entire file
	ReadLine()	Reads a line of the file up to the newline character
	Skip()	Skips the specified number of characters
	SkipLine()	Skips the next line when reading the file
	Write()	Writes a string to the file
	WriteBlankLines()	Writes the specified number of blank lines to the file
	WriteLine()	Writes a string followed by a newline character to the file
Property	AtEndOfLine	Returns true if the pointer is immediately before the end-of-line marker

Type	Item	Description
	AtEndOfStream	Returns true if the pointer is at the end of the file
	Column	Returns the column number of the current pointer position
	Line	Returns the line number of the current pointer position

Example

Listing 14.96 creates an example of the `TextStream` object, and then writes a string to the file. After the string has been written, the stream is closed. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.96 *Creating an Instance of the `TextStream` Object*

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject")

// Create a TextStream object.
var myTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt", 2, true)

// Write a string to the file.
myTextStream.Write("Hello, World!");

// Close the stream to the file.
myTextStream.Close();
```

TextStream.AtEndOfLine

JScript2.0+

Syntax

```
textstream.AtEndOfLine
```

Description

The `AtEndOfLine` property of the `TextStream` object returns `true` when the end of a line in a text line is found.

Example

Listing 14.97 opens two files. The first file is for reading a string from, whereas the second file is written to—one character at a time—as the script iterates through. The result is that the first line of the file opened for reading is written to the second file. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.97 Using the `AtEndOfLine` Property to Find the End of the First Line in a File

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
    ➤ 1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
    ➤ 2, true);

// Iterate through the file one character at a time and write it
// to the second file.
while(!myInputTextStream.AtEndOfLine){
    myOutputTextStream.Write(myInputTextStream.Read(1));
}

// Close the stream to the files.
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.AtEndOfStream

JScript2.0+

Syntax

`textstream.AtEndOfStream`

Description

The `AtEndOfStream` property of the `TextStream` object returns true when the end of a file in a text file is found.

Example

Listing 14.98 opens two files. The first file is for reading a string from, whereas the second file is written to—one character at a time—as the script iterates through. The result is that the file opened for reading is written to the second file. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.98 Using the `AtEndOfStream` Property to Find the End of the File

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
    ➤ 1, true);
```



```
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt", "  
    ↗2, true);  
  
// Iterate through the file one character at a time and write it  
// to the second file.  
while(!myInputTextStream.AtEndOfStream){  
    myOutputTextStream.Write(myInputTextStream.Read(1));  
}  
  
// Close the stream to the files.  
myInputTextStream.Close();  
myOutputTextStream.Close();
```

TextStream.Close()

JScript2.0+

Syntax

```
textstream.Close()
```

Description

The Close() method of the TextStream object closes the previously opened file.

Example

Listing 14.99 shows you how to open and close a text file. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.99 Using the Close() Method to Close a Text File

```
// Create a FileSystemObject object.  
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");  
  
// Create a TextStream object.  
var myTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt", 1, true);  
  
// Close the stream to the file.  
myTextStream.Close();
```

TextStream.Column

JScript2.0+

Syntax

```
textstream.Column
```

Description

The Column property of the TextStream object returns the column number of the current pointer position.

Example

Listing 14.100 opens a file for reading and a file for writing. The script iterates through each character of the first file, and then writes it and the column number to the second file. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.100 *Using the Column Property*

```
// Create a FileSystemObject object
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
                                                    ➤1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ➤2, true);

// Iterate through the file one character at a time and writes it
// to the second file. Includes the column number with each character.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.Column + ": ");
    myOutputTextStream.WriteLine(myInputTextStream.Read(1));
}

// Close the stream to the files
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.Line

JScript2.0+

Syntax

textstream.Line

Description

The Line property of the TextStream object returns the line number of the current pointer position.

Example

Listing 14.101 opens a file for reading and a file for writing. The script iterates through each character of the first file, and then writes it and the line number to the second file. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.101 Using the Line Property

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
                                                    ↪1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ↪2, true);

// Iterate through the file one character at a time and write it
// to the second file. Include the line number with each character.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.Line + " : ");
    myOutputTextStream.WriteLine(myInputTextStream.Read(1));
}

// Close the stream to the files
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.Read()

JScript2.0+

Syntax

```
textstream.Read(num)
```

Description

The `Read()` method of an instance of the `TextStream` object reads the *num* characters passed to the method starting at the current pointer position.

Example

Listing 14.102 opens two files. The first file is for reading a string from, whereas the second file is written to—one character at a time—as the script iterates through. The result is that the file opened for reading is written to the second file. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.102 Using the Read() Method

[illegible]

Listing 14.102 Continued

```
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ↵2, true);

// Iterate through the file one character at a time and write it
// to the second file.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.Read(1));
}

// Close the stream to the files.
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.ReadAll()

JScript2.0+

Syntax

```
textstream.ReadAll()
```

Description

The ReadAll() method of an instance of the TextStream object reads the entire text file. Be careful when using this method with large files because it will use up a good amount of system memory.

Example

Listing 14.103 opens two files. The first file is for reading a string from, whereas the second file is written to. The result is that the file opened for reading is written to the second file. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.103 Using the ReadAll() Method

```
// Create a FileSystemObject object
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
                                                    ↵1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ↵2, true);

// Iterate through the file one character at a time and write it
// to the second file.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.ReadAll());
}
```

```
// Close the stream to the files.  
myInputTextStream.Close();  
myOutputTextStream.Close();
```

TextStream.ReadLine()

JScript2.0+

Syntax

textstream.ReadLine()

Description

The ReadLine() method of an instance of the TextStream object reads a line from a text file up to a newline character. Be careful when using this method with large files because it will use up a good amount of system memory.

Example

Listing 14.104 opens two files. The first file is for reading a string from, whereas the second file is written to. Each line is read and written one at a time. The result is that the file opened for reading is written to the second file. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.104 Using the ReadLine() Method

```
// Create a FileSystemObject object.  
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");  
  
// Create a TextStream object to read from and one to write to.  
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",  
                                                    ➤1, true);  
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",  
                                                    ➤2, true);  
  
// Iterate through the file one line at a time and write it  
// to the second file.  
while(!myInputTextStream.AtEndOfStream){  
    myOutputTextStream.WriteLine(myInputTextStream.ReadLine());  
}  
  
// Close the stream to the files.  
myInputTextStream.Close();  
myOutputTextStream.Close();
```

TextStream.Skip()

JScript2.0+

Syntax

textstream.Skip(*num*)

Description

The `Skip()` method of an instance of the `TextStream` object skips *num* characters.

Example

Listing 14.105 opens two files. The first file is for reading a string from, whereas the second file is written to. The `Skip()` method is used to skip the first five characters before the file is read. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.105 Using the `Skip()` Method

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
                                                    ↵1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ↵2, true);

// The Skip method is used to skip the first 5 characters.
myInputTextStream.Skip(5);

// Iterate through the file one character at a time and write it
// to the second file.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.Read(1));
}

// Close the stream to the files.
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.SkipLine()

JScript2.0+

Syntax

```
textstream.SkipLine()
```

Description

The `SkipLine()` method of an instance of the `TextStream` object skips the next line.

Example

Listing 14.106 opens two files. The first file is for reading a string from, whereas the second file is written to. The `SkipLine()` method is used to skip the first line before

the file is read. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.106 Using the SkipLine() Method

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject");

// Create a TextStream object to read from and one to write to.
var myInputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt",
                                                    ➤1, true);
var myOutputTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test2.txt",
                                                    ➤2, true);

// The SkipLine method is used to skip the first line of the input file.
myInputTextStream.SkipLine();

// Iterate through the file one character at a time and write it
// to the second file.
while(!myInputTextStream.AtEndOfStream){
    myOutputTextStream.Write(myInputTextStream.Read(1));
}

// Close the stream to the files.
myInputTextStream.Close();
myOutputTextStream.Close();
```

TextStream.Write()

JScript2.0+

Syntax

```
textstream.Write(string)
```

Description

The Write() method of an instance of the TextStream object is used to write a string to the text file.

Example

Listing 14.107 opens a text file and writes a string to it. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.107 Using the Write() Method

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject")
```

Listing 14.107 Continued

```
// Create a TextStream object.
var myTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt", 2, true)

// Write a string to the file.
myTextStream.Write("Hello, World!");

// Close the stream to the file.
myTextStream.Close();
```

TextStream.WriteBlankLines()

JScript2.0+

Syntax

```
textstream.WriteBlankLines(num)
```

Description

The WriteBlankLines() method of an instance of the TextStream object is used to write *num* newline characters to the text file.

Example

Listing 14.108 opens a text file and writes a string to it. The WriteBlankLines() method is then used to place a newline before a second string is written. This file can be executed with the Windows Script Host (wscript.exe or cscript.exe).

Listing 14.108 Using the WriteBlankLines() Method

```
// Create a FileSystemObject object.
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject")

// Create a TextStream object.
var myTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt", 2, true)

// Write the first string to the file.
myTextStream.Write("Hello, World!");

// Write a newline character to the file.
myTextStream.WriteBlankLines(1);

// Write a second string to the file.
myTextStream.Write("Hello, World Again!");

// Close the stream to the file.
myTextStream.Close();
```


TextStream.WriteLine()

JScript2.0+

Syntax

```
textstream.WriteLine()
```

Description

The `WriteLine()` method of an instance of the `TextStream` object is used to write a string followed by a newline to the text file.

Example

Listing 14.109 opens a text file and writes a string followed by a newline to it. This file can be executed with the Windows Script Host (`wscript.exe` or `cscript.exe`).

Listing 14.109 Using the WriteLine() Method

```
// Create a FileSystemObject object
var myFileSysObj = new ActiveXObject("Scripting.FileSystemObject")

// Create a TextStream object
var myTextStream = myFileSysObj.OpenTextFile("c:\\temp\\test.txt", 2, true)

// Write a string followed by a newline to the file
myTextStream.WriteLine("Hello, World!");

// Close the stream to the file
myTextStream.Close();
```




ON THE CD-ROM

CHAPTER 15

Windows Script Host

This chapter contains all the objects, properties, and methods that make up Microsoft's Windows Script Host as it relates to JScript. Most books that cover Windows Script Host only cover it from the VBScript perspective, but any scripting language can be used. Because this is a JavaScript book and JavaScript is a great scripting language, we will approach this chapter strictly from the JScript (JavaScript) perspective.

The chapter is in alphabetical order, by objects, to provide you with quick, easy access to the major parts of the language. The properties and methods of every object appear alphabetically after the respective parent object, using the simple dot notation. For example, the `Echo()` method is found after the `WScript` object as `WScript.Echo()`.

NOTE

As of the writing of this book, Windows Script Host 5.6 was in beta release, so the new 5.6 material covered in this chapter is subject to some changes in the final release of 5.6. The newest Windows Script Host engine and various associated documents can be downloaded from Microsoft's scripting page (www.msdn.microsoft.com/scripting).

Scripting.Signer

WSH 5.6+

Syntax

Scripting.Signer

Description

The `Scripting.Signer` object allows script authors to sign their scripts with a digital signature. Similarly, the object allows a recipient of the script to verify the signature's authenticity and trustworthiness. The `Scripting.Signer` object requires a valid certificate, used to decode or encode the digital signature, in order to be of any use.

Example

Listing 15.1 uses the `Scripting.Signer` object to try to sign a script. If no certificate or file is specified, the user is alerted.

Listing 15.1 *Using the `Scripting.Signer` Object*

```
<job>
<script language="JScript">
    var Signer, File, Cert, Store;

    //If no certificate or file exists then display the usage of the signer.
    if (!(WScript.Arguments.Named.Exists("cert") &&
        WScript.Arguments.Named.Exists("file")))
    {
        WScript.Echo("No certificate or file");
        WScript.Quit();
    }

    //Create a Scripting.Signer object.
    Signer = new ActiveXObject("Scripting.Signer");

    File = WScript.Arguments.Named("file");
    Cert = WScript.Arguments.Named("cert");
    Store = WScript.Arguments.Named("store");

    //Sign the file.
    Signer.SignFile(File, Cert, Store);

</script>
</job>
```

NOTE

The `<job>` and `</job>` tags are used to define the beginning and end of a job within a Windows Script file. If the `<job>` tag contains the argument `"id="` followed by a user-defined name, the `id` can be used to execute a particular job within a Windows Script file, if multiple jobs exist in the file. To execute a particular job, simply add the argument `"Job"` followed by the job ID to the `cscript.exe` command.

WScript

WSH 1.0+

Syntax

WScript

Description

The WScript object provides access to properties of the Windows Script Host (WScript.exe or cscript.exe) that is currently running. The object also provides methods to create and read objects. Table 15.1 lists all the properties and methods associated with the WScript object.

Table 15.1 Arguments and Methods Associated with WScript Object

Type	Item	Description
Properties	Arguments	Collection of command-line parameters
	FullName	Full path to host executable file
	Interactive	Mode of the script
	Name	Name of the Windows Script Host
	Path	Directory where Windows Script Host executable resides
	ScriptFullName	Full path to script
	ScriptName	Name of script
	StdErr	Provides access to error stream
	Stdin	Provides access to input stream
	Stdout	Provides access to output stream
	Version	Version of Windows Script Host
	WshArguments	Object containing command-line parameters
Methods	ConnectObject()	Connects objects to event functions
	CreateObject()	Creates new objects
	DisconnectObject()	Disconnects objects from event functions
	Echo()	Display strings
	GetObject()	Gets an object from memory or from a file
	Quit()	Stop script execution
	Sleep()	Pauses execution

In addition to the methods and properties shown in the table, WScript is also a parent to some objects. Specifically, the CreateObject() method can be used to create any one of the following objects as a child of the WScript object: WshController, WshNetwork, WshShell.

Example

In Listing 15.2, the Echo() method of the WScript object is used to display contents of the WScript's Name property.

Listing 15.2 Using the WScript Object

```
<job>
<script language="JScript">

WScript.Echo("WSH Name=",WScript.Name);

</script>
</job>
```

WScript.Arguments

WSH 1.0+

Syntax

WshObject.Arguments

Description

The Arguments property of a WScript object contains a pointer to the WshArguments object, which is a collection that contains the command-line parameters associated with the WScript object.

Example

Listing 15.3 uses the Arguments property to retrieve a collection of arguments that were passed into the script. For example, assume that the script in Listing 15.1 was saved in a Windows script file called *myscript.wsf* and was executed in a MS-DOS window using the following line:

```
cscript.exe myscript.wsf car truck boat plane
```

In this case, the arguments that are passed into the script are *car*, *truck*, *boat*, and *plane*. These arguments are retrieved from within the script using the Arguments property and displayed as shown in Figure 15.1.

Listing 15.3 Retrieving Arguments Using the Arguments Property

```
<job>
<script language="JScript">

if(WScript.Arguments.Count())
{
    WScript.Echo("Arguments:");

    //Get collection of arguments.
    WSOobjectArgs = WScript.Arguments

    //Loop through all the arguments in collection.
    for(i=0; i<WSOobjectArgs.Count(); i++)
    {
```



```

        WScript.Echo(i + " = " + WScriptArgs(i));
    }
}
else
{
    WScript.Echo("No arguments");
}
</script>
</job>

```

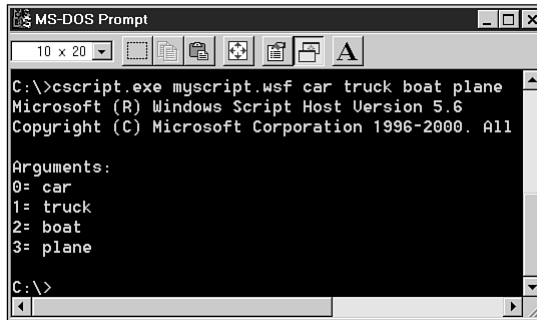


Figure 15.1

Accessing command-line parameters with the Arguments property.

WScript.ConnectObject()

WSH 2.0+

Syntax

WshObject.ConnectObject(*theObject*, prefixString)

Description

The ConnectObject() method connects the specified object's (*theObject*) event sources to functions located in the script. When the object fires an event with this new connection in place, the Windows Script Host will call a specified function. The prefixString argument is used to form the name of the function. The name of the subroutine is formed by combining the prefixString with the name of the event. For example if prefixString is "MyObject" and the event is "OnBegin", the function name would be "MyObjectOnBegin".

Example

Listing 15.4 demonstrates how to use the ConnectObject() method using a fictitious object called BogusObject. An instance of the BogusObject is created and then used in the call to the ConnectObject() method. Should a "BogusEvent" event occur while

executing the `myMethod()` method, the `MyBogusFunctionBogusEvent()` function will be called. Since this program uses fictitious objects and events it is not meant to be executed but rather just analyzed.

Listing 15.4 Using the `ConnectObject()` Method

```
<job>
<script language="JScript">

//Create a bogus object
MyBogusObject = WScript.CreateObject("BogusObject");

//Connect object to a function
WScript.ConnectObject(MyBogusObject, "MyBogusFunction");

MyBogusObject.myMethod();

//Handle the BogusEvent event
function MyBogusFunctionBogusEvent()
{
    //handle the event
}

//Disconnect the object from the function.
WScript.DisconnectObject(MyBogusObject);

</script>
</job>
```

WScript.CreateObject()

WSH 2.0+

Syntax

```
WshObject.CreateObject(programID, [prefixString])
```

Description

The `CreateObject()` method creates an object specified by the program ID (*programID*) of the object to create. An optional argument can be provided to make the Windows Script Host connect the new object's outgoing interface to the script file. If the new object fires an event with this connection in place, the Windows Script Host will call a subroutine located in the script. The name of the subroutine is formed by combining the second argument (*prefixString*) with the name of the event. For example if *prefixString* is "MyObject" and the event is "OnBegin", the subroutine name would be "MyObjectOnBegin".

Example

Listing 15.5 uses the `CreateObject()` method to create a `WshNetwork` object called `MyWshNetwork`. The new `MyWshNetwork` object is then used to display the name of the computer.

Listing 15.5 *Creating a `WshNetwork` Object with the `CreateObject()` Method*

```
<job>
<script language="JScript">

MyWshNetwork = WScript.CreateObject("WScript.Network");
WScript.echo(MyWshNetwork.computerName);

</script>
</job>
```

WScript.DisconnectObject()

WSH 2.0+

Syntax

```
WshObject.DisconnectObject(theObject)
```

Description

The `DisconnectObject()` method disconnects the specified object's (*theObject*) event sources from the Windows Script Host.

Example

Listing 15.4, shown in an earlier entry, demonstrates how to use the `DisconnectObject()` method using a fictitious object called `MyBogusObject`.

WScript.Echo()

WSH 1.0+

Syntax

```
WshObject.Echo(string1, string2, ..., stringN )
```

Description

The `Echo()` method displays the specified strings separated by spaces. If the script is being run from `WScript.exe`, the string will be displayed in a dialog box. If the script is run with the Console Script Host (`CScript.exe`), the string will be displayed in the console from which the script is run. Under `CScript.exe`, a newline character is placed at the end of the output after every call to the `Echo()` method. If no string is provided when calling the method, a blank line will be displayed.

Example

Listing 15.6 uses the `Echo()` method to display some various strings. If the script is run from `WScript.exe`, the two calls to the `Echo()` method will appear in two separate dialog boxes. If the script is run from `CScript.exe`, the output from each call to the `Echo()` method will result in two lines of text.

Listing 15.6 *Using the `Echo()` Method*

```
<job>
<script language="JScript">

color = "red";

WScript.Echo("Apples are", color);
WScript.Echo("Bananas are not", color);

</script>
</job>
```

WScript.FullName

WSH 2.0+

Syntax

`WshObject.FullName`

Description

The `FullName` property contains the full path to the host executable file.

Example

The `FullName` property is used in Listing 15.7 to display the full path to the Windows Script Host executable file. For example, if `WScript.exe` were located in the `C:\WINDOWS` directory, the `FullName` property would contain `C:\WINDOWS\WSCRIPT.EXE`.

Listing 15.7 *Displaying the `FullName` Property*

```
<job>
<script language="JScript">

WScript.Echo("The FullName=", WScript.FullName);

</script>
</job>
```

WScript.GetObject()

WSH 2.0+

Syntax

```
WshObject.GetObject( fileName, [programID], [prefixString] )
```

Description

The `GetObject()` method retrieves an existing object from memory or creates a new one from a file. To create an object from a file, simply provide the full path to the file (*fileName*). To retrieve an object from memory, simply specify the program ID (*programID*). If the object to be retrieved is registered as a single-instance object, only one instance of the object is created.

An optional argument can be provided to make the Windows Script Host connect the new object's outgoing interface to the script file. If the new object fires an event with this connection in place, the Windows Script Host will call a subroutine located in the script. The name of the subroutine is formed by combining the second argument (*prefixString*) with the name of the event. For example if *prefixString* is "MyObject" and the event is "OnBegin", the subroutine name would be "MyObjectOnBegin".

NOTE

The `GetObject()` method can be used with all COM classes, independent of the language used to create the object.

The `GetObject()` method cannot be used to obtain a reference to a Microsoft Visual Basic class created with Visual Basic 4.0 or earlier.

Example

The following fictitious code snippet uses the `GetObject()` method to create a `channel` object from the `MYSONG.MID` file that is associated with the `MYSOUNDPGM` application. This would cause the `MYSOUNDPGM` application to start and open the object `CHANNEL` from within the `MYSONG` file.

```
channel = GetObject("C:\SONGS\MYSONG.MID", "MYSOUNDPGM.CHANNEL");
```

WScript.Interactive

WSH 5.6+

Syntax

```
WshObject.Interactive
```

Description

The `Interactive` property contains the mode of the script in the form of a Boolean value. The property can also be used to set the mode. The mode can be either batch or interactive, which is represented by 0 and 1, respectively.

In *interactive* mode (1), the script can interact with users through dialog boxes. In *batch* mode (0) input or output cannot take place, so there is no interaction between the script and users.

NOTE

The mode of a script can also be set using the Cscript.exe command line switches `//I` (for Interactive), and `//B` (for Batch).

Example

In Listing 15.8, the script mode is retrieved and displayed using the Interactive property.

Listing 15.8 Retrieving the Script Mode with the Interactive Property

```
<job>
<script language="JScript">

WScript.Echo("Script Mode=",WScript.Interactive);

</script>
</job>
```

WScript.Name

WSH 2.0+

Syntax

`WshObject.Name`

Description

The Name property contains the name of the Windows Script Host (WSH).

Example

In Listing 15.9 the Windows Script Host name is retrieved and displayed using the Name property.

Listing 15.9 Retrieving the WSH Name with the Name Property

```
<job>
<script language="JScript">

WScript.Echo("WSH Name=",WScript.Name);

</script>
</job>
```

WScript.Path

WSH 2.0+

Syntax

WshObject.Path

Description

The Path property contains the name of the directory in which the WScript.exe or CScript.exe resides.

Example

In Listing 15.10, the directory in which the Windows Script Host executable is located is retrieved and displayed using the Path property.

Listing 15.10 Retrieving the WSH Directory with the Path Property

```
<job>
<script language="JScript">

WScript.Echo("WSH Path=",WScript.Path);

</script>
</job>
```

WScript.Quit()

WSH 2.0+

Syntax

WshObject.Quit (*exitCode*)

Description

The Quit() method causes the execution of the script to stop immediately. If you need the script to return an integer exit code, simply provide an optional *exitCode* parameter to the method call.

Example

In Listing 15.11, the Quit() method is called just before the second Echo() method is reached. This causes execution of the script to stop, which prevents the second Echo() method to be executed.

Listing 15.11 Stopping with the Quit() Method

```
<job>
<script language="JScript">
```

Listing 15.11 Continued

```
WScript.Echo("Before the call to the Quit() method.");  
WScript.Quit();  
WScript.Echo("After the call to the Quit() method.");  
  
</script>  
</job>
```

WScript.ScriptFullName

WSH 2.0+

Syntax

WshObject.ScriptFullName

Description

The ScriptFullName property contains the full path to the script that is currently being executed.

Example

In Listing 15.12, the ScriptFullName property is used to display the full path to the script that is being executed.

Listing 15.12 Displaying the ScriptFullName Property

```
<job>  
<script language="JScript">  
  
WScript.Echo(WScript.ScriptFullName);  
  
</script>  
</job>
```

WScript.ScriptName

WSH 2.0+

Syntax

WshObject.ScriptName

Description

The ScriptName property contains the name of the script file that is currently being executed. No path information is contained in this property.

Example

In Listing 15.13, the ScriptName property is used to display the name of the script file that is being executed.

Listing 15.13 *Displaying the ScriptName Property*

```
<job>
<script language="JScript">

WScript.Echo(WScript.ScriptName);

</script>
</job>
```

WScript.Sleep()WSH 2.0+

Syntax*WshObject.Sleep(milliseconds)***Description**

The `Sleep()` method causes the script to pause execution for the number of milliseconds specified by the *milliseconds* parameter.

Example

In Listing 15.14, the `Sleep()` method pauses execution of the script for 10 seconds (10,000 milliseconds). Using the `Sleep()` method causes a 10 second pause between when the string above the `Sleep()` method is displayed and when the string below the `Sleep()` method is displayed.

Listing 15.14 *Pausing Script Execution with the Sleep() Method*

```
<job>
<script language="JScript">

WScript.Echo("Before the Sleep() method.");
WScript.Sleep(10000);
WScript.Echo("After the Sleep() method.");

</script>
</job>
```

WScript.StdErrWSH 2.0+

Syntax*WshObject.StdErr***Description**

The `StdErr` property provides access to the write-only error output stream of the current script.

CAUTION

The `StdErr` property can only be accessed using the Console Script Host (CScript.exe). Accessing the property with the Windows Script Host (WScript.exe) will cause an error.

Example

In Listing 15.15, the error stream is assigned to the variable *standardError* using the `StdErr` property. A string is then sent to the standard error output using the `write()` method. In order for this script to work, it must be executed using the Console Script Host (CScript.exe). Unless you have redirected the standard error output, the string will appear in the console.

Listing 15.15 Sending a String to the Standard Error Output

```
<job>
<script language="JScript">

standardError = WScript.StdErr;
standardError.write("This string was sent to the standard error output");

</script>
</job>
```

WScript.StdIn

WSH 2.0+

Syntax

WshObject.StdIn

Description

The `StdIn` property provides access to the read-only input stream of the current script.

CAUTION

The `StdIn` property can only be accessed using the Console Script Host (CScript.exe). Accessing the property with the Windows Script Host (WScript.exe) will cause an error.

Example

In Listing 15.16, the standard input stream is assigned to the variable *standardInput* using the `StdIn` property. The user is then prompted to enter some text. The text is then read in using the `Read()` method, and the first 10 characters entered by the user are stored in the `str` variable and then displayed in the console.

Listing 15.16 *Entering Text Using Standard Input*

```
<job>
<script language="JScript">

standardIn = WScript.StdIn;

WScript.Echo("Please enter some text:");
str = standardIn.read(10);
WScript.Echo("You Entered:",str);

</script>
</job>
```

WScript.StdOutWSH 2.0+

Syntax*WshObject*.StdOut**Description**

The StdOut property provides access to the write-only standard output stream of the current script.

 CAUTION

The StdOut property can only be accessed using the Console Script Host (CScript.exe). Accessing the property with the Windows Script Host (WScript.exe) will cause an error.

Example

In Listing 15.17, the standard output stream is assigned to the variable *standardOutput* using the StdOut property. A string is then sent to standard output using the *write()* method. In order for this script to work, it must be executed using the Console Script Host (CScript.exe). Unless you have redirected standard output, the string will appear in the console.

Listing 15.17 *Sending a String to the Standard Output*

```
<job>
<script language="JScript">

standardOutput = WScript.StdOut;
standardOutput.write("This string was sent to the standard output");

</script>
</job>
```

WScript.Version

WSH 1.0+

Syntax

WshObject.Version

Description

The Version property contains the version of the Windows Script Host that is being used.

Example

In Listing 15.18, the version of the Windows Script Host is displayed using the Version property.

Listing 15.18 *Displaying the WSH Version Using the Version Property*

```
<job>
<script language="JScript">

WScript.Echo("WSH Version:",WScript.Version);

</script>
</job>
```

WScript.WshArguments

WSH 5.6+

Syntax

WshObject.WshArguments

Description

The WshArguments property of a WScript object contains a pointer to the WshArguments object, which contains the command-line parameters associated with the WScript object.

Example

Listing 15.19 uses the Arguments property to retrieve a collection of arguments that were passed into the script. For example, assume that the script from Listing 15.1 was saved in a Windows script file called `myscript.wsf` and was executed in a MS-DOS window using the following line:

```
cscript.exe myscript.wsf car truck boat plane
```

In this case the arguments that are passed into the script are `car`, `truck`, `boat`, and `plane`. These arguments are retrieved from within the script using the Arguments property and displayed as shown earlier in Figure 15.1.

Listing 15.19 Retrieving Arguments Using the Arguments Property

```
<job>
<script language="JScript">

if(WScript.WshArguments.Count())
{
    WScript.Echo("Arguments:");

    //Get collection of arguments
    WSOBJECTArgs = WScript.WshArguments

    //Loop through all the arguments in object
    for(i=0; i<WSObjectArgs.Count(); i++)
    {
        WScript.Echo(i + " = " + WSOBJECTArgs(i));
    }
}
else
{
    WScript.Echo("No arguments");
}
</script>
</job>
```

WScript.WshControllerWSH 5.6+

Syntax

None

Description

The WshController object is a child object of WScript. Because it is not directly accessible through the WScript.WshController syntax, see this chapter's entry for WshController for more information.

WScript.WshNetworkWSH 5.6+

Syntax

None

Description

The WshNetwork object is a child object of WScript. Because it is not directly accessible through the WScript.WshNetwork syntax, see this chapter's entry for WshNetwork for more information.

WScript.WshShell

WSH 5.6+

Syntax

None

Description

The WshShell object is a child object of WScript. Because it is not directly accessible through the WScript.WshShell syntax, see this chapter's entry for WshShell for more information.

WshArguments

WSH 2.0+

Syntax

ArgObj = WScript.Arguments

WScript.Arguments(*number*)

Description

The WshArguments object provides access to the command-line parameters associated with the WScript object. The WshArguments object is accessed from the WScript object's Arguments property and is in fact a child of the WScript object. Table 15.2 lists all the properties and methods associated with the WshArguments object.

The contents of the WshArguments object can also be accessed directly by placing a *number* in parenthesis directly after the WshArgument object's name.

Table 15.2 Arguments and Methods Associated with the WshArguments Object

Type	Item	Description
Properties	Count	Number of parameters (VBScript only)
	Length	Number of parameters
	Named	Object of only named parameters
	Unnamed	Object of only unnamed parameters
Methods	Count()	Returns number of parameters
	Item()	Provides access to specific parameters
	ShowUsage()	Displays the usage information associated with script.

In addition to the methods and properties shown in the table, WshArguments is also a parent to some other objects. Specifically, the Named and Unnamed properties provide access to the WshNamed and WshUnnamed objects.

Example

The code in Listing 15.20 creates a `WshArgument` object, called `WshArgsObj`, from the `WScript.Arguments` property. If the `WshArgsObj` object contains any parameters, they are accessed directly from the object using parenthesis. So, if you executed this code using the command `cscript.exe listing15-20.wsf Chris Craig`, the arguments `Chris` and `Craig` would be displayed.

Listing 15.20 *Displaying the Contents of the `WshArguments` Object*

```
<job>
<script language="JScript">

//Get collection of arguments.
WshArgsObj = WScript.Arguments;

if(WshArgsObj.Count())
{
    WScript.Echo("Arguments:");

    //Loop through all the arguments in collection.
    for(i=0; i< WshArgsObj.Count(); i++)
    {
        WScript.Echo(i + " = " + WshArgsObj(i));
    }
}
else
{
    WScript.Echo("No arguments");
}
</script>
</job>
```

WshArguments.Count

WSH 2.0+

Syntax

`WshArgsObj.Count`

Description

The `Count` property contains the number of parameters that were passed into the Windows Script Host. This number includes both the named and unnamed parameters.

NOTE

The `Count` property is only available in VBScript. Use the `Length` property to access the number of parameters in JScript.

Example

No example is provided because this property is not available in JScript.

WshArguments.Count()

WSH 2.0+

Syntax

```
WshArgsObj.Count()
```

Description

The Count() method returns the number of parameters that were passed into the Windows Script Host. This number includes both the named and unnamed parameters.

Example

The code in Listing 15.21 creates a WshArgument object, called *WshArgsObj*, from the WScript.Arguments property. The Count() method is then used to display the number of parameters passed into the Windows Script Host. For example, if the code were run in MS-DOS using the line "cscript.exe listing21.wsf red blue", the Count() method would return the number 2.

Listing 15.21 *Displaying the Contents of the Count() Method*

```
<job>
<script language="JScript">

//Get collection of arguments
WshArgsObj = WScript.Arguments;

//Display the number of parameters
WScript.Echo("Number of parameters: ", WshArgsObj.Count());

</script>
</job>
```

WshArguments.Item()

WSH 2.0+

Syntax

```
WshArgsObj.Item(number)
```

Description

The Item() method is used to access a single item in the WshArguments collection. The item (WScript parameter) to be retrieved is specified with the *number* argument.

Example

The code in Listing 15.22 creates a WshArgument object, called *WshArgsObj*. If the *WshArgsObj* object contains any parameters, they are accessed using the Item() method.

Listing 15.22 Displaying WScript Parameters Using the Item() Method

```
<job>
<script language="JScript">

//Get collection of arguments
WshArgsObj = WScript.Arguments;

if(WshArgsObj.Count())
{
    WScript.Echo("Arguments:");

    //Loop through all the arguments in collection
    for(i=0; i< WshArgsObj.Count(); i++)
    {
        WScript.Echo(i + " = " + WshArgsObj.Item(i));
    }
}
else
{
    WScript.Echo("No arguments");
}
</script>
</job>
```

WshArguments.lengthWSH 2.0+

Syntax*WshArgsObj.length***Description**

The Length property contains the number of parameters that were passed into the Windows Script Host. This number includes both the named and unnamed parameters.

Example

The code in Listing 15.23 creates a WshArgument object, called *WshArgsObj*. The Length property is then used to determine if the object contains any parameters. If the *WshArgsObj* object does contain parameters, the Length property is used again to set up a loop in which all the parameters are displayed.

Listing 15.23 Utilizing the Length Property

```
<job>
<script language="JScript">

//Get collection of arguments
WshArgsObj = WScript.Arguments;
```

Listing 15.23 Continued

```
if(WshArgsObj.Length)
{
    WScript.Echo("Arguments:");

    //Loop through all the arguments in collection.
    for(i=0; i< WshArgsObj.Length; i++)
    {
        WScript.Echo(i + " = " + WshArgsObj.Item(i));
    }
}
else
{
    WScript.Echo("No arguments");
}
</script>
</job>
```

WshArguments.Named

WSH 5.6+

Syntax*WshArgsObj*.Named**Description**

The Named property contains the WshNamed collection object. The WshNamed object contains parameters of the WshArguments object with the unnamed parameters removed. A named parameter is one that is made up of a key/value pair in which the key and value are separated with a colon (:) and the key is preceded by a forward slash (/).

Example

The code in Listing 15.24 creates a WshArgument object, called *WshArgsObj*. The Named property is then used to access the WshNamed object, which is used to display the value of the color parameter if such a parameter should exist. For example let us assume that the script was run using the following line:

```
cscript.exe listing24.wsf /color:green
```

Then the string "color= green" would be displayed.

Listing 15.24 Using the Named Property to Display the Color Parameter

```
<job>
<script language="JScript">

//Get collection of arguments.
WshArgsObj = WScript.Arguments;
```

```
//Get collection of named arguments.
WshNamedArgsObj = WshArgsObj.Named

if(WshNamedArgsObj.length)
{
    WScript.Echo("color=", WshNamedArgsObj.Item("color"));
}
else
{
    WScript.Echo("No named arguments");
}
</script>
```

WshArguments.ShowUsage()

WSH 5.6+

Syntax

```
WshArgsObj.ShowUsage()
```

Description

The ShowUsage() method displays a help screen for the script. This help screen displays information about the script's command-line options. This information is defined in the runtime portion of the script as defined within the <runtime> and </runtime> tags.

Example

Listing 15.25 contains a script that displays the color the user entered on the command line when executing the script. If no arguments are specified on the command line, the ShowUsage() method is called to display the usage information contained in the <runtime> tags. Figure 15.2 shows the usage information that is displayed.

Listing 15.25 *Using the ShowUsage() Method to Display Usage Information*

```
<job>
<runtime>

<description>
This script displays the color you pass in
</description>
  <named
    name = "color"
    helpstring = "Your favorite color"
    type = "string"
    required = "true"
  />
  <example>Example: listing25.wsf /color:purple</example>
</runtime>
```

Listing 15.25 Continued

```
<script language="JScript">

//Get collection of arguments
WshArgsObj = WScript.Arguments;

if(WshArgsObj.Named.length == 0)
{
    //Display how to use script
    WshArgsObj.ShowUsage();
}
else
{
    WScript.Echo("Your favorite color is", WshArgsObj.Named("color"));
}

</script>
</job>
```

**Figure 15.2**

Displaying the usage information for the script.

WshArguments.Unnamed

WSH 5.6+

Syntax

WshArgsObj.Unnamed

Description

The Unnamed property contains the WshUnnamed collection object. The WshUnnamed object contains parameters of the WshArguments object with the named parameters removed. A named parameter is one that is made up of a key/value pair in which the key and value are separated by a colon (:) and the key is preceded by a forward slash (/).

Example

The code in Listing 15.26 creates a WshArgument object, called *WshArgsObj*. The Unnamed property is then used to access the WshUnnamed object, which is used to

display the unnamed parameters if such parameters should exist. For example, let us assume that the script was run using the following line:

```
cscript.exe listing26.wsf /color:green blue red
```

When the code is executed using the command the unnamed parameters blue and red are displayed.

Listing 15.26 Using the Unnamed Property to Display Unnamed Parameters

```
<job>
<script language="JScript">

//Get collection of arguments
WshArgsObj = WScript.Arguments;

//Get collection of unnamed arguments
WshUnnamedArgsObj = WshArgsObj.Unnamed

if(WshUnnamedArgsObj.length)
{
    WScript.Echo("Unnamed Arguments:");

    //Loop through all the arguments in unnamedcollection
    for(i=0; i< WshUnnamedArgsObj.Length; i++)
    {
        WScript.Echo(i + " = " + WshUnnamedArgsObj.Item(i));
    }
}
else
{
    WScript.Echo("No unnamed arguments");
}
</script>
</job>
```

WshArguments.WshNamed

WSH 5.6+

Syntax

None

Description

WshNamed object is a child object of WshArguments. Because it is not directly accessible through the WshArguments.WshNamed syntax, see this chapter's entry for WshNamed for more information.

WshArguments.WshUnnamed

WSH 5.6+

Syntax

None

Description

WshUnnamed object is a child object of WshArguments. Because it is not directly accessible through the WshArguments.WshUnnamed syntax, see this chapter's entry for WshUnnamed for more information.

WshController

WSH 5.6+

Syntax

```
WshControllerObj = WScript.CreateObject("WScript.Controller")
```

Description

The WshController object is a child of the WScript object. The WshController object allows remote scripts, WshRemote objects, to be created using its CreateScript() method. To create the WshController object, you must use the WScript's CreateScript() method.

NOTE

The WshController object is not available on Windows 9x operating systems.

Example

Listing 15.27 uses the CreateObject() method to create a WshController object that is a child of the WScript object. The new WshController object is then used to create a remote script that is executed.

Listing 15.27 Creating and Using a WshController Object

```
<job>
<script language="JScript">

// Create a WshController object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\windows\myscript.wsf");

// Start the script.
remoteScript.Execute();
```

```
</script>
</job>
```

WshController.CreateScript()

WSH 5.6+

Syntax

```
WshControllerObj.CreateScript(commandLine, machine)
```

Description

The `CreateScript()` method creates a `WshRemote` object that allows scripts on one computer system to be executed remotely on another machine. The full pathname of the remote script, as well as any command-line options, are passed in to the *commandLine* argument. The script file can be on a completely different computer system.

If the script is to be executed on a remote machine, the name of the remote computer should be passed into the *machine* argument using the *Universal Naming Convention (UNC)*. The *machine* argument is optional, so if it is left blank, the script is run on the controller's computer system.

CAUTION

The `CreateScript()` method sets up a script to run and establishes a connection with a remote computer system, but the script is not actually started until a call is made to the `Execute()` method of the `WshRemote` object.

NOTE

The `CreateScript()` method is not available on Windows 9x operating systems.

Example

Listing 15.28 creates a `WshController` object, called *WshCtrlObj* using the `CreateObject()` method. A remote script located at "C:\scripts\myScript.wsf" is created using the `CreateScript()` method. The remote script, called *myRemoteScript*, is executed using the `Execute()` method.

Listing 15.28 Using the `CreateObject` Method to Create a Remote Script

```
<job>
<script language="JScript">

//Create Controller object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

//Create the remote script
myRemoteScript = WshCtrlObj.CreateScript("c:\\scripts\\myScript.wsf");
```

Listing 15.28 Continued

```
//Execute the remote script
myRemoteScript.Execute();

</script>
</job>
```

WshController.WshRemote

WSH 5.6+

Syntax

None

Description

WshRemote object is a child object of WshController. Because it is not directly accessible through the WshController.WshRemote syntax, see this chapter’s entry for WshRemote for more information.

WshEnvironment

WSH 2.0+

Syntax

```
WshEnvObj(string)
```

Description

The WshEnvironment object provides access to the entire set of environment variables. This object is accessed through the Environment property of the WshShell object. Depending on the settings of the Environment property, the WshEnvironment object will contain “User”, “System”, “Volatile”, or “Process” related environment variables. For more information on creating the WshEnvironment object, see the “WshShell.Environment” section in this chapter.

Table 15.3 lists all the properties and methods associated with the WshEnvironment object. In addition to the methods shown in this array, an environment variable can be directly accessed from the WshEnvironment object by simply placing an environment variable’s name in parenthesis directly after the name of the WshEnvironment object.

Table 15.3 Arguments and Methods Associated with the WshEnvironment Object

Type	Item	Description
Properties	Length	Contains number of environment variables
	Count()	Returns number of environment variables
	Item()	Provides access to specific environment variable
Method	Remove()	Removes an environment variable

Table 15.4 lists some of the environment variables that are available in the Windows operating system with a short description. The table also shows each variable's availability based on the Environment property setting (P=Process, S=System, U=User). The "Volatile" setting supports none of the variables in the table.

Table 15.4 Windows Environment Variables

Name	P	S	U	Description
NUMBER_OF_PROCESSORS	X	X		Number of processors running on machine.
PROCESSOR_ARCHITECTURE	X	X		Processor type of user's machine.
PROCESSOR_IDENTIFIER	X	X		Processor ID of user's machine.
PROCESSOR_LEVEL	X	X		Processor level of user's machine.
PROCESSOR_REVISION	X	X		Processor version of user's machine.
OS	X	X		OS on user's machine.
COMSPEC	X	X		Executable file for command prompt (usually cmd.exe).
HOMEDRIVE		X		Primary local drive.
HOMEPATH		X		Default users directory.
PATH	X	X	X	Path.
PATHEXT	X			Extensions for executable files.
PROMPT	X			Command prompt.
SYSTEMDRIVE	X			Local drive where the system directory is located.
SYSTEMROOT	X			System directory. Same as WINDIR.
WINDIR	X	X		System directory. Same as SYSTEMROOT.
TEMP	X		X	Directory for temporary files.
TMP	X		X	Directory for temporary files.

Example

A WshEnvironment object is created and used to access all the "Process" environment variables in Listing 15.29. The new WshEnvObj is used to display the value of the "WINDIR" environment variable. When this script is executed, the path to your system directory will be displayed.

Listing 15.29 Displaying the System Directory Path with the WshEnvironment Object

```
<job>
<script language="JScript">

//Create a shell object
var WshShellObj = WScript.CreateObject("WScript.Shell");
```

Listing 15.29 Continued

```
//Retrieve the WshEnvironment object
var WshEnvObj = WshShellObj.Environment("Process");

//Display the System directory
WScript.Echo("System directory=", WshEnvObj("WINDIR"));

</script>
</job>
```

WshEnvironment.Count()

WSH 2.0+

Syntax

```
WshEnvObj.Count()
```

Description

The Count() method returns the number of environment variables that are in the WshEnvironment object. Depending on the settings of the Environment property on creation, the WshEnvironment object will contain different numbers of environment variables. For more information on the Environment property, see the “WshShell.Environment” entry in this chapter.

Example

Listing 15.30 uses the Count() method to display the number of "Process" related environment variables that are in the WshEnvironment object.

Listing 15.30 Displaying the number of Environment Variables Using the Count() Method

```
<job>
<script language="JScript">

//Create a shell object
var WshShellObj = WScript.CreateObject("WScript.Shell");

//Retrieve the WshEnvironment object
var WshEnvObj = WshShellObj.Environment("Process");

//Display the number of environment variables in object
WScript.Echo("Count=", WshEnvObj.Count());

</script>
</job>
```

WshEnvironment.Item()

WSH 2.0+

Syntax

WshEnvObj.Item(*string*)

Description

The `Item()` method is used to access the value of a single environment variable from `WshEnvironment` object. The name of the environment variable to retrieve is specified by the *string* argument.

Example

A `WshEnvironment` object is created and used to access all the "Process" environment variables in Listing 15.31. The new *WshEnvObj* and its `Item()` method are used to display the value of the "WINDIR" environment variable. When this script is executed, the path to your system directory will be displayed.

Listing 15.31 *Displaying the System Directory Path with the* `Item()` *Method*

```
<job>
<script language="JScript">

//Create a shell object.
var WshShellObj = WScript.CreateObject("WScript.Shell");

//Retrieve the WshEnvironment object.
var WshEnvObj = WshShellObj.Environment("Process");

//Display the System directory
WScript.Echo("System directory=", WshEnvObj.Item("WINDIR"));

</script>
</job>
```

WshEnvironment.length

WSH 2.0+

Syntax

WshEnvObj.length

Description

The `length` property contains the number of environment variables that are in the `WshEnvironment` object. Depending on the settings of the `Environment` property on

creation, the `WshEnvironment` object will contain different numbers of environment variables. For more information on the `Environment` property, see the “`WshShell.Environment`” entry in this chapter.

Example

Listing 15.32 uses the `length` property to display the number of “Process” related environment variables that are in the `WshEnvironment` object.

Listing 15.32 Displaying the Number of Environment Variables Using the `length` Property

```
<job>
<script language="JScript">

//Create a shell object
var WshShellObj = WScript.CreateObject("WScript.Shell");

//Retrieve the WshEnvironment object
var WshEnvObj = WshShellObj.Environment("Process");

//Display the number of environment variables in object
WScript.Echo("Length=", WshEnvObj.length);

</script>
</job>
```

WshEnvironment.Remove()

WSH 2.0+

Syntax

```
WshEnvObj.Remove(envVar)
```

Description

The `Remove()` method allows you to delete environment variables by simply specifying the name of the environment variable in the method. Environment variables that are removed with this method are only removed for the current session.

NOTE

An Environment variable must exist in the `WshEnvironment` object in order to be deleted. The environment variables that are in the `WshEnvironment` object depend on the settings of the `Environment` property when the `WshEnvironment` object is created.

Example

Listing 15.33 begins by checking to see if the WINDIR environment variable exists. If it does, the value stored in the variable is displayed. The variable is then deleted using the Remove() method. Finally, a success message is displayed if the environment variable is successfully deleted.

Listing 15.33 *Deleting an Environment Variable with the Remove() Method*

```
<job>
<script language="JScript">

//Create a shell object
var WshShellObj = WScript.CreateObject("WScript.Shell");

//Retrieve the WshEnvironment object
var WshEnvObj = WshShellObj.Environment("Process");

if(WshEnvObj.Item("WINDIR"))
{
    //Display the number of environment variables in object
    WScript.Echo("WINDIR=", WshEnvObj.Item("WINDIR"));

    //Delete environment variable
    WshEnvObj.Remove("WINDIR");

    //Was deletion successful?
    if(!WshEnvObj.Item("WINDIR"))
    {
        WScript.Echo("WINDIR successfully deleted!");
    }
}
else
{
    WScript.Echo("WINDIR was not deleted!");
}
}
else
{
    WScript.Echo("WINDIR does not exist so it cannot be deleted!");
}

</script>
</job>
```

WshNamed

WSH 5.6+

Syntax

WshNamed

Description

The WshNamed object provides access to the collection or arguments in which the WshArguments.Named property returns. Table 15.5 lists all the properties and methods associated with the WshNamed object.

Table 15.5 Arguments and Methods Associated with the WshNamed Object

Type	Item	Description
Properties	Item	Access Items
	Length	The number of items in the Arguments collection
Methods	Count()	The number of switches in the WshNamed or WshUnnamed object
	Exists()	Indicates whether key value exists

Example

Listing 15.34 uses the Arguments property to retrieve a collection of arguments that were passed into the script and assigns that to a new object named WshNamed. Then a loop is run to print out all the arguments to the screen. Suppose that the script was named myScript.wsf and was run with the following command in DOS:

```
cscript myScript.wsf "Where is Tilly" "The Dog is outside"
```

After executing the script, the two arguments “Where is Tilly” and “The Dog is outside” would be displayed to the user.

Listing 15.34 Using the WshNamed Object

```
<job>
<script language="JScript">

// Create a WshNamed object
WshNamed = WScript.Arguments;

// Loop through the arguments and echo each argument
for(i=0; i<WshNamed.Count(); i++){
    WScript.Echo("The Argument is: " + WshNamed.Item(i) );
}

</script>
</job>
```

WshNamed.Count()

WSH 5.6+

Syntax

WshNamed.Count()

Description

The `Count()` method of the `WshNamed` object returns an integer value representing the number of switches.

Example

Listing 15.35 uses the `Arguments` property to retrieve a collection of arguments that were passed into the script and assigns that to a new object named `WshNamed`. Then the number of arguments is printed to the screen. Suppose that the script was named `myScript.wsf` and was run with the following command in DOS:

```
cscript myScript.wsf "What are you doing?" "Reading Pure JavaScript!"
```

The number of arguments (2) would be displayed to the screen.

Listing 15.35 Retrieving the Number of Arguments Using `Count`

```
<job>
<script language="JScript">

// Create a WshNamed object
WshNamed = WScript.Arguments;

// Print out the number of arguments
WScript.Echo("The Number of arguments is: " + WshNamed.Count() );

</script>
</job>
```

WshNamed.Exists()

WSH 5.6+

Syntax

WshNamed.Exists(value)

Description

The `Exists()` method is used to determine whether a specific value exists in the `WshNamed` object. A boolean value of `true` or `false` is returned.

Example

Listing 15.36 uses the `Arguments` property to retrieve a collection of arguments that were passed into the script and assigns that to a new object named `WshNamed`. Then it

checks for the specific value “c” in the argument list. Suppose that the script was named `myScript.wsf` and was run with the following command in DOS:

```
cscript myScript.wsf /c "Where is the Doc"
```

A message indicating that the key was found would be displayed.

Listing 15.36 Using the Exists() Method of WshNamed

```
<job>
<script language="JScript">

// Creating WshNamed object
WshNamed = WScript.Arguments.Named;

// If the /c switch is used, then a message
// indicating that the key was found is displayed.
if( WshNamed.Exists("C") )
{
    WScript.Echo("The Key was found");
}

</script>
</job>
```

WshNamed.Item

WSH 5.6+

Syntax

WshNamed.Item(key)

Description

The `Item` property is used to gain access to each item in the `WshNamed` object.

Example

Listing 15.37 uses the `Arguments` property to retrieve a collection of arguments that were passed into the script and assigns that to a new object named `WshNamed`. Then a loop is run to print out all the arguments to the screen. Suppose that the script was named `myScript.wsf` and was run with the following command in DOS:

```
cscript myScript.wsf "Where is Big Money Jlai?" "He is out riding his bike"
```

After execution, each argument, "Where is Big Money Jlai?" and "He is out riding his bike", would be displayed to the screen.

Listing 15.37 Displaying the Arguments with the Items Property

```
<job>
<script language="JScript">
```



```
// Create the WshNamed object
WshNamed = WScript.Arguments;

// Loop through and print out each argument
for(i=0; i<WshNamed.Count(); i++){
    WScript.Echo("The Argument Item is: " + WshNamed.Item(i) );
}

</script>
</job>
```

WshNamed.length

WSH 5.6+

Syntax

WshObject.length

Description

The length property returns the number of command-line parameters for the script. This is a read-only property and is essentially the number of items in the arguments collection.

Example

Listing 15.38 uses the Arguments property to retrieve a collection of arguments that were passed into the script and assigns that to a new object named *WshNamed*. Then the length property is used to display the length to the screen. Suppose that the script was named *myScript.wsf* and was run with the following command in DOS:

```
cscript myScript.wsf "Angela" "Pete"
```

The length representing the number of arguments would be displayed to the screen.

Listing 15.38 *Displaying the Arguments with the length Property*

```
<job>
<script language="JScript">

// Creating WshNamed object
WshNamed = WScript.Arguments;

WScript.Echo("The Length is: " + WshNamed.Length );

</script>
</job>
```

WshNetwork

WSH 2.0+

Syntax

WshNetwork

Description

The WshNetwork object provides access to the Microsoft Windows Network. This encompasses all network items such as drives and printers. Table 15.6 lists all the properties and methods associated with the WshNetwork object.

Table 15.6 *Properties and Methods Associated with the WshNetwork Object*

Type	Item	Description
Properties	ComputerName	Name of the Windows machine
	UserDomain	User domain of local machine
	UserName	Name registered in Windows
Methods	AddPrinterConnection()	Maps remote printer
	AddWindowsPrinterConnection()	Adds Printer Connection
	EnumNetworkDrives()	Gets all networked drives
	EnumPrinterConnection()	Gets all printer connections
	MapNetworkDrive()	Maps a network drive
	RemoveNetworkDrive()	Removes a network drive
	RemovePrinterConnection()	Removes a printer
	SetDefaultPrinter()	Sets the Default printer

Example

In Listing 15.39, the Echo() method of the WScript object is used to display contents of the WshNetwork's UserName property.

Listing 15.39 *Using the WshNetwork Object*

```
<job>
<script language="JScript">

// Create a new WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Print out the username
WScript.Echo("The UserName for this machine is: ",WshNetwork.UserName);

</script>
</job>
```

WshNetwork.AddPrinterConnection()

WSH 2.0+

Syntax

WshNetwork.AddPrinterConnection(localName, remoteName, updateProfile, userName, password)

Description

The `AddPrinterConnection()` method is used to map a remote printer to a specified machine. The *localName* of the printer should be supplied as the first parameter. The *remoteName* of the printer should be specified as the second parameter. The *updateProfile* is a boolean value that specifies whether the mapping should be stored in the user profile (default if `false`). For printers that require login authentication to be added, the *userName* and *password* should be supplied. The *updateProfile*, *userName*, and *password* are optional parameters.

Example

Listing 15.40 shows an example of how to map a printer to the local machine.

Listing 15.40 Using the `AddPrinterConnection()` Method to Map a Remote Printer

```
<job>
<script language="JScript">

// Create a new WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Define the printer to be added
var printer = "\\printers\DefaultPrinter";
var localName = "myPrinter";

// Add the printer to the network
WshNetwork.AddPrinterConnection(localName, printer);

</script>
</job>
```

WshNetwork.AddWindowsPrinterConnection()

WSH 2.0+

Syntax

WshObject.AddWindowsPrinterConnection(*printerPath*, *driveName*, *printerPort*)

Description

The `AddWindowsPrinterConnection()` method is used to add printer connections to the windows environment. The *printerPath* parameter specifies the path to the printer and must be specified. The *driverName* specifies the name of the driver to use and is

ignored on the Windows NT/2000 environment. The *printerPort* specifies the port on which the printer connection should be made and is optional. The default value is LPT1 and is ignored on Windows NT/2000.

Example

Listing 15.41 shows an example of how to add the Epson Stylus 850 printer to the current Windows environment.

Listing 15.41 *Using the* AddWindowsPrinterConnection *Method*

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Define the printer to be added
var printerPath = "\\printers\\DefaultPrinter";
var driver = "Epson Stylus 850";

// Add the network printer
WshNetwork.AddWindowsPrinterConnection(printerPath, driver);

</script>
</job>
```

WshNetwork.ComputerName

WSH 2.0+

Syntax

WshNetwork.ComputerName

Description

The ComputerName property returns a string representation of the computer name.

Example

Listing 15.42 shows an example of how to use the ComputerName property to get the current computer's name.

Listing 15.42 *Using the* ComputerName *Property*

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");
```

```
// Print out the computer name
WScript.Echo("The name of this computer is: " + WshNetwork.ComputerName);

</script>
</job>
```

WshNetwork.EnumNetworkDrives()

WSH 2.0+

Syntax

`WshNetwork.EnumNetworkDrives()`

Description

The `EnumNetworkDrives()` method is used to obtain the current network drive mappings.

Example

In Listing 15.43 the network drives are retrieved and displayed to the user within a for loop.

Listing 15.43 *Using the `EnumNetworkDrives()` Method to Retrieve the Network Drive Mappings*

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");
var myDrives = WshNetwork.EnumNetworkDrives();

// Loop through the drives and print out each drive number
for (i=0; i<myDrives.Count(); i++){
    WScript.Echo("Drive Number " + i + " is " + myDrives.Item(i+1) );
}

</script>
</job>
```

WshNetwork.EnumPrinterConnection()

WSH 2.0+

Syntax

`WshNetwork.EnumPrinterConnection()`

Description

The `EnumPrinterConnection()` method is used to obtain the current network printer mappings.

Example

In Listing 15.44 the list of network printers is displayed.

Listing 15.44 *Using the EnumPrinterConnection() Method to Display the Network Printers*

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");
var myPrinters = WshNetwork.EnumPrinterConnections();

// Loop through the network printers and list each printer
for (i=0; i<myPrinters.Count(); i++){
    WScript.Echo("Printer " + myPrinters.Item(i) + " is " +
    ➤myPrinters.Item(i+1));
}

</script>
</job>
```

WshNetwork.MapNetworkDrive()

WSH 2.0+

Syntax

```
WshNetwork.MapNetworkDrive(driveLetter, driveName, updateProfile,
➤userName, password)
```

Description

The `MapNetworkDrive()` method is used to map a drive from the network to your local machine. The *driveLetter* and the *driveName* must be supplied. The *driveLetter* pertains to the letter on which the drive will be on the local machine. The *driveName* represents the name of the network drive to map. The *updateProfile* is a boolean value that specifies whether the mapping should be stored in the user profile (default is false). For mappings that require login authentication, the *userName* and *password* should be supplied.

Example

Listing 15.45 shows how the “everyone” drive can be mapped to the local machine as the “T” drive.

Listing 15.45 *Using the MapNetworkDrive() Method to Map a Network Drive*

```
<job>
<script language="JScript">
```

```
// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Map a network drive
WshNetwork.MapNetworkDrive("T:", "\\some_network\everyone");

</script>
</job>
```

WshNetwork.RemoveNetworkDrive()

WSH 2.0+

Syntax

```
WshNetwork.RemoveNetworkDrive(driveName, bForce, bUpdateProfile)
```

Description

The `RemoveNetworkDrive()` method is used to remove a current network drive from the local machine. The *driveName* must be specified as a string representing the local name or the remote name of the drive to be removed. The *bForce* is a boolean value (default is `false`) that, if supplied and set to `true`, will force a removal of the drive whether it is in use or not. The *bUpdateProfile* is a boolean value (default is `false`), which if specified, will remove the mapping from the user profile. *bForce* and *bUpdateProfile* are optional parameters.

Example

In Listing 15.46, a network drive is mapped and then removed using the `RemoveNetworkDrive` method.

Listing 15.46 Using the `RemoveNetworkDrive()` Method to Remove a Network Drive

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Map a network drive
WshNetwork.MapNetworkDrive("T:", "\\some_network\everyone");

// Remove the mapped network drive
WshNetwork.RemoveNetworkDrive("T");

</script>
</job>
```

WshNetwork.RemovePrinterConnection()

WSH 2.0+

Syntax

```
WshNetwork.RemovePrinterConnection(printer, bForce, bUpdateProfile)
```

Description

The `RemovePrinterConnection()` method is used to remove a specified printer connection from the local machine. The name of the printer to be removed is specified in the *printer* parameter. The *bForce* is a boolean value (default is `false`) that, if supplied and set to `true`, will force a removal of the drive whether it is in use or not. The *bUpdateProfile* is a boolean value (default is `false`), which if specified, will remove the mapping from the user profile. *bForce* and *bUpdateProfile* are optional parameters.

Example

In Listing 15.47, the printer connected to LPT1 is removed from the local machine.

Listing 15.47 *Using the RemovePrinterConnection() Method to Remove a Network Printer*

```
<job>
<script language="JScript">

// Create a new WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Remove the printer on LPT1
WshNetwork.RemovePrinterConnection("LPT1:");

</script>
</job>
```

WshNetwork.SetDefaultPrinter()

WSH 2.0+

Syntax

```
WshNetwork.setDefaultPrinter(printerName)
```

Description

The `setDefaultPrinter()` method is used to set the default printer to a specified remote printer, *printerName*.

Example

In Listing 15.48, the default printer is set to the “blueroom” printer.

Listing 15.48 Using the `setDefaultPrinter()` Method to Set the Default Printer

```
<job>
<script language="JScript">

// Create a WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Set the default printer
WshNetwork.SetDefaultPrinter("\\domainname\blueroom");

</script>
</job>
```

WshNetwork.UserDomain

WSH 2.0+

Syntax

`WshObject.UserDomain`

Description

The `UserDomain` property is used to obtain a string representation of the user domain name.

Example

Listing 15.49 shows how to display the user domain name to the user.

Listing 15.49 Using the `UserDomain` Property to Display the User Domain Name

```
<job>
<script language="JScript">

// Create a new WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Print the domain name
WScript.Echo("The User Domain name is: " + WshNetwork.UserDomain);

</script>
</job>
```

WshNetwork.UserName

WSH 2.0+

Syntax

`WshObject.UserName`

Description

The `UserName` property is used to get a string representation of the username.

Example

Listing 15.50 shows how to use the `UserName` property to display the username.

Listing 15.50 *Using the `UserName` property to Display the Username*

```
<job>
<script language="JScript">

// Create WshNetwork object
var WshNetwork = new ActiveXObject("Wscript.Network");

// Display the username
WScript.Echo("The User name is: " + WshNetwork.UserName);

</script>
</job>
```

WshRemote

WSH 5.6+

Syntax

WshRemote

Description

The `WshRemote` object provides access to a remote script process. It is created by using the `CreateScript()` method. Table 15.7 lists all the properties and methods associated with the `WshRemote` object.

Table 15.7 *Arguments and Methods Associated with the `WshRemote` Object*

Type	Item	Description
Properties	Error	Returns the <code>WshRemoteError</code> object
	Status	Provides status information for the script
Methods	Execute()	Begins execution of the remote script
	Terminate()	Terminates the remote script
Events	End	Signals the remote script to end execution
	Error	Signals the remote script to end execution when an error occurs
	Start	Signals the remote script to begin execution

Example

In Listing 15.51, a `WshController` object is used to create a `WshRemote` object. Then the remote script `hello.wsf` is executed using the `Execute` method.

 NOTE

The WshController object is not available on Windows 9x operating systems.

Listing 15.51 Using the WshRemote Object

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

</script>
</job>
```

WshRemote.End

WSH 5.6+

Syntax

WshRemote.End

Description

The End event is used to signal to the remote script to end execution.

Example

In Listing 15.52 the remote script, hello.wsf, is executed. Upon ending, a message is displayed indicating that execution has finished.

Listing 15.52 Using the End Event of the WshRemote Object

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();
```

Listing 15.52 Continued

```
// If the end event occurs, then return finished message.
if(remoteScript.End){
    WScript.Echo("Finished execution");
}
</script>
</job>
```

WshRemote.Error (Event)

WSH 5.6+

Syntax

WshObject.Error

Description

The Error event is used to signal the scripting engine to end execution of a remote script when the remote script stops because of an error.

Example

In Listing 15.53 the remote script, hello.wsf, is executed. Upon an error, a message is displayed indicating that execution has terminated due to an error.

Listing 15.53 Using the Error Event to Check for Remote Script Errors

```
<job>
<script language="JScript">

// Create a WshController object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// If Error event occurs, then display error message.
if(remoteScript.Error){
    WScript.Echo("Script Ended Due To Error");
}

</script>
</job>
```

WshRemote.Error (Property)

WSH 5.6+

Syntax

WshRemote.Error

Description

The *Error* property is used to obtain information in relation to an error, which causes a remote script to terminate prematurely. The *Error* property returns a *WshRemoteError* object.

Example

In Listing 15.54, the error description is displayed from the *showErr* function if an error occurred in the remote script.

Listing 15.54 Using the Error Property to Display the Error Description

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle any errors which may occur.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The WshRemote.Error is: " + myError.Description);
    WScript.Quit(-1);
}

</script>
</job>
```

WshRemote.Execute()

WSH 5.6+

Syntax

WshRemote.Execute()

Description

The *Execute()* method is used to start execution of a remote script. This is not the same as the *WScript.Exec()* method.

Example

Listing 15.55 shows how to use the `Execute` method to start the remote script `hello.wsf`.

Listing 15.55 *Starting the Remote Script with* `Execute`

```
<job>
<script language="JScript">

// Create a WshController object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

</script>
</job>
```

WshRemote.WshRemoteError

WSH 5.6+

Syntax

WshRemote.WshRemoteError

Description

The `WshRemoteError` object is a child object of `WshRemote`. Because it is not directly accessible through the *WshRemote.WshRemoteError* syntax, see this chapter's entry for `WshRemoteError` for more information.

WshRemote.Start

WSH 5.6+

Syntax

WshRemote.Start

Description

The `Start` event signals the scripting engine to begin execution of the remote script. It is fired when the `Execute` method is called.

Example

Listing 15.56 shows how the `Start` event is used to indicate when the script has started execution.

Listing 15.56 Using the Start Event of WshRemote

```
<job>
<script language="JScript">

// Create a WshController object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// If the Start event is detected, then display message.
if(remoteScript.Start){
    WScript.Echo("Remote script Started Execution");
}

</script>
</job>
```

WshRemote.Status

WSH 5.6+

Syntax

WshRemote.Status

Description

The Status property is used to obtain information on the status of a script run with the Exec method. The status property is only used when a script is run asynchronously. It will return one of two values. If WshRunning (equivalent to 0) is returned, the script is still running. If WshFinished (equivalent to 1) is returned, the script has finished running.

Example

Listing 15.57 uses the Status property to see whether the script is finished running.

Listing 15.57 Using the Status Property to Check the Script Status

```
<job>
<script language="JScript">

// Create a WshController object
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");
```

```
// Start the script.
remoteScript.Execute();

// Check the Status property. Depending on the value,
// an appropriate message is displayed.
if (remoteScript.Status == 0){
    WScript.Echo("The Script is still Running");
}

if (remoteScript.Status == 1){
    WScript.Echo ("The Script is finished");
}

</script>
</job>
```

WshRemote.Terminate()

WSH 5.6+

Syntax

```
WshRemote.Terminate()
```

Description

The `Terminate()` method is used to stop execution of a remote script. No value is returned from the method.

Example

Listing 15.58 uses the `Terminate` method to stop the execution of the `hello.wsf` script.

Listing 15.58 Using the `Terminate` Method to Terminate Execution of a Remote Script

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Terminate remote script.
remoteScript.Terminate();

</script>
</job>
```


WshRemoteError

WSH 5.6+

Syntax

WshRemoteError

Description

The WshRemoteError object is used to obtain error information when a remote script is terminated due to an error. It is returned by the Error property of the WshRemote object. Table 15.8 lists all the properties associated with the WshRemoteError object.

Table 15.8 Properties Associated with WshRemoteError Object

Type	Item	Description
Properties	Character	Character in the line of code that contains an error
	Description	Description of an error
	Line	Line of code in which an error occurred
	Number	Error number
	Source	COM object responsible for causing an error
	SourceText	Source code line that caused an error

Example

The code in Listing 15.59 creates a WshRemoteError object called *myError*.

Listing 15.59 Creating a WshRemoteError Object

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

function showErr(){
    // Create WshRemoteError object.
    var myError = remoteScript.Error;
}

</script>
</job>
```

WshRemoteError.Character

WSH 5.6+

Syntax

WshRemoteError.Character

Description

The Character property returns a signed long integer that specifies the character in the line of source code in which the error occurred.

Example

Listing 15.60 shows how to use the Character property to display the exact character in the source code where the error occurred.

Listing 15.60 Using the Character Property to Display the Error Character

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle errors in the remote script. If an error
// occurs, then the character in which the error occurred is displayed.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The remote error is at character: " + myError.Character);
    WScript.Quit(-1);
}

</script>
</job>
```

WshRemoteError.Description

WSH 5.6+

Syntax

WshRemoteError.Description

Description

The Description property returns a string containing a brief description of the error that occurred.

Example

Listing 15.61 uses the Description property to display the error description.

Listing 15.61 *Using the Description Property to Display the Remote Error Description*

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle remotescript errors. If error occurs, then
// the error description is displayed.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The remote error description is : " + myError.Description);
    WScript.Quit(-1);
}

</script>
</job>
```

WshRemoteError.Line

WSH 5.6+

Syntax

WshRemoteError.Line

Description

The Line property returns an unsigned long integer indicating the line in which the error occurred.

Example

Listing 15.62 uses the Line property of the WshRemoteError object to display the line in the source code in which the error occurred.

Listing 15.62 Using the Line Property to Display the Error Line Number

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle remotescript errors. If error occurs, then
// the error line is displayed.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The remote error is at line: " + myError.Line);
    WScript.Quit(-1);
}

</script>
</job>
```

WshRemoteError.Number

WSH 5.6+

Syntax

WshRemoteError.Number

Description

The Number property returns an unsigned long integer representing the error number of the script error.

Example

Listing 15.63 uses the Number property of the WshRemoteError object to display the number of the error that occurred.

Listing 15.63 Using the Number Property to Display the Remote Error Number

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");
```

```
// Start the script.
remoteScript.Execute();

// Function to handle remotescript errors. If error occurs, then
// the error number is displayed.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The remote error Number is: " + myError.Number);
    WScript.Quit(-1);
}

</script>
</job>
```

WshRemoteError.Source

WSH 5.6+

Syntax

WshRemoteError.Source

Description

The Source property returns a string representing the COM object responsible for causing the script error.

Example

The code in Listing 15.64 uses the Source property of the WshRemoteError object to get the COM object that caused the error in the remote script.

Listing 15.64 *Using the Source Property to Display the Remote Error COM Source Object*

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle remotescript errors. If error occurs, then
// the source of the error is displayed.
function showErr(){
```

Listing 15.64 Continued

```
var myError = remoteScript.Error;
WScript.Echo("The remote error was caused by: " + myError.Source);
WScript.Quit(-1);
}

</script>
</job>
```

WshRemoteError.SourceText

WSH 5.6+

Syntax

WshRemoteError.SourceText

Description

The SourceText property returns a string containing the line of source code in which the error occurred.

Example

Listing 15.65 uses the SourceText property of the WshRemoteError object to display the source code text of the line in which the error occurred.

Listing 15.65 Using the SourceText Property to Display the Remote Error Source Code Line

```
<job>
<script language="JScript">

// Create a WshController object.
WshCtrlObj = WScript.CreateObject("WScript.Controller");

// Create a WshRemote object.
remoteScript = WshCtrlObj.CreateScript("c:\My Documents\hello.wsf");

// Start the script.
remoteScript.Execute();

// Function to handle remotescript errors. If error occurs, then
// a string representing the source code for the error is displayed.
function showErr(){
    var myError = remoteScript.Error;
    WScript.Echo("The remote error was caused by: " + myError.SourceText);
    WScript.Quit(-1);
}

</script>
</job>
```

WshScriptExec

WSH 5.6+

Syntax

WshScriptExec

Description

The WshScriptExec object provides status and access to the stdIn, stdOut, and stderr streams for the object returned from the WshShell.Exec() method. Table 15.9 lists all the properties and methods associated with the WshScriptExec object.

Table 15.9 *Properties and Methods Associated with the WshScriptExec Object*

Type	Item	Description
Properties	Status	Status of script run with Exec
	StdErr	Standard Error stream of script
	StdIn	Standard Input stream of script
	StdOut	Standard Output stream or script
Methods	Terminate()	Ends execution of script started with the Exec method

Example

Listing 15.66 shows how to create a new WshScriptExec object.

Listing 15.66 *Creating a New WshScriptExec Object*

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//create a WshScriptExec object.
WshScriptExec = WshShell.Exec("cscript c:\my documents\hello.wsf");

</script>
</job>
```

WshScriptExec.Status

WSH 5.6+

Syntax

WshScriptExec.Status

Description

The Status property provides access to status information on scripts run from the Exec method. The Status property is only used when a script is run asynchronously. It will

return one of two values. If `WshRunning` (equivalent to 0) is returned, the script is still running. If `WshFinished` (equivalent to 1) is returned, the script has finished running.

Example

The code in Listing 15.67 uses the `Status` property to see whether the script is finished running or not. Depending on the status code returned, a message is displayed to the user.

Listing 15.67 *Using the `Status` Property to Get a Script's Status*

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//create a WshScriptExec object
WshScriptExec = WshShell.Exec("cscript c:\my documents\hello.wsf");

// Check the Status of the script. Depending on it's value,
// an appropriate message displayed.
if (WshScriptExec.Status == 0){
    WScript.Echo("The Script is still running");
}

if (WshScriptExec.Status == 1){
    WScript.Echo ("The Script is finished");
}

</script>
</job>
```

WshScriptExec.StdErr

WSH 5.6+

Syntax

`WshScriptExec.StdErr`

Description

The `StdErr` property provides access to the `stderr` output stream for the `Exec` object. The `Exec` object does not need to be running a script.

CAUTION

The `StdErr` property can only be accessed using the Console Script Host (`CScript.exe`). Accessing the property with the Windows Script Host (`WScript.exe`) will cause an error.

Example

Listing 15.68 uses the error stream that is assigned to the variable `standardError` using the `StdErr` property. A string is then sent to a standard error using the `write()` method. In order for this script to work, it must be executed using the Console Script Host (CScript.exe). Unless you have redirected standard error output, the string will appear in the console.

Listing 15.68 Using the `StdErr` Property of `WshScriptExec`

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//create a WshScriptExec object
WshScriptExec = WshShell.Exec("cscript c:\my documents\hello.wsf");

// Set the standard error stream
standardError = WshScriptExec.StdErr;

// Write text to stderr.
standardError.write("This string was sent to the standard error output");

</script>
</job>
```

WshScriptExec.StdIn

WSH 5.6+

Syntax

`WshScriptExec.StdIn`

Description

The `StdIn` property provides access to the read-only `stdin` input stream of the `Exec` object.

CAUTION

The `StdIn` property can only be accessed using the Console Script Host (CScript.exe). Accessing the property with the Windows Script Host (WScript.exe) will cause an error.

Example

In Listing 15.69 the standard input stream is assigned to the variable `standardInput` using the `StdIn` property. The user is then prompted to enter some text. The text is then read in using the `Read()` method, and the first 10 characters entered by the user are stored in the `str` variable and then displayed in the console.

Listing 15.69 Using the StdIn Property of WshScriptExec

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//Create a WshScriptExec object.
WshScriptExec = WshShell.Exec("cscript c:\my documents\hello.wsf");

// Set the standard Input property
standardInput = WshScriptExec.StdIn;

WScript.Echo("Please enter some text:");

// Read the input in StdIn
str = standardInput.read(10);

// Echo what was read from StdIn
WScript.Echo("You entered:",str);

</script>
</job>
```

WshScriptExec.StdOut

WSH 5.6+

Syntax

WshScriptExec.Stdout

Description

The StdOut property provides a write-only stdout output stream of the Exec object.

CAUTION

The StdOut property can only be accessed using the Console Script Host (CScript.exe). Accessing the property with the Windows Script Host (WScript.exe) will cause an error.

Example

In Listing 15.70 the standard output stream is assigned to the variable *standardOutput* using the StdOut property. A string is then sent to standard output using the *write()* method. In order for this script to work, it must be executed using the Console Script Host (CScript.exe). Unless you have redirected standard output, the string will appear in the console.

Listing 15.70 Using the StdOut Property of WshScriptExec

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//Create a WshScriptExec object.
WshScriptExec = WshShell.Exec("cscript c:\my documents\hello.wsf");

// Set the standard output stream
standardOutput = WshScriptExec.StdOut;

// Write text to StdOut stream
standardOutput.write("This string was sent to the standard output");

</script>
</job>
```

WshScriptExec.Terminate()

WSH 5.6+

Syntax

```
WshScriptExec.Terminate()
```

Description

The `Terminate()` method is used to terminate the process started by the `Exec` method.

Example

Listing 15.71 shows how the `hello.wsf` script is stopped by the `Terminate` method.

Listing 15.71 Terminating a Script with the Terminate Method

```
<job>
<script language="JScript">

var WshShell = new ActiveXObject("WScript.Shell");

//create a WshScriptExec object
aScript = WshShell.Exec("cscript c:\my documents\hello.wsf");

//Terminate script
aScript.Terminate();

</script>
</job>
```

WshShell

WSH 2.0+

Syntax

WshShell

Description

The WshShell object is used to create new processes, shortcuts, and access environment variables. Table 15.10 lists all the properties and methods associated with the WshShell object.

Table 15.10 *Properties and Methods Associated with WshShell Object*

Type	Item	Description
Property	CurrentDirectory	Returns a string with the fully qualified path of the current working directory
	Environment	Returns a WshEnvironment object
	SpecialFolders	Returns a SpecialFolders object
	WshShortcut	Creates a child object
	WshUrlShortcut	Creates a child object
	WshEnvironment	Creates a child object
	WshSpecialFolders	Creates a child object
Method	AppActivate()	Activates an application window
	CreateShortcut()	Creates a new shortcut or opens an existing shortcut
	Exec()	Runs an application in the child command shell
	ExpandEnvironmentStrings()	Returns an environment variable's expanded value
	LogEvent()	Adds an event entry to the log file
	Popup()	Displays text in a pop-up box
	RegDelete()	Delete a key from the registry
	RegRead()	Returns a value of the key from the registry
	RegWrite()	Creates a new key or alters an existing key in the registry
	Run()	Runs a program in the new process
	SendKeys()	Sends keystrokes to the active window

Example

Listing 15.72 shows how to create a new instance of the WshShell object.

Listing 15.72 Using the WshShell Object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

</script>
</job>
```

WshShell.AppActivate()

WSH 2.0+

Syntax

WshShell.AppActivate(app)

Description

The AppActivate() method of the WshShell object activates a specified application window. The application name (can be a window title or application task ID) is specified as the parameter *app*. The method returns a boolean value that indicates whether the procedure call was successful or not.

Example

Listing 15.73 uses the WshShell AppActivate() method to activate a notepad application window. (This example assumes that the notepad application is currently running.)

Listing 15.73 Activating the Application Window Using AppActivate()

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Activate Notepad.
WshShellObj.AppActivate("Notepad");

</script>
</job>
```

WshShell.CreateShortcut()

WSH 2.0+

Syntax

WshShell.CreateShortcut(str)

Description

The `CreateShortcut()` method of the `WshShell` object is used to create a new shortcut or open an existing shortcut. A single parameter, *str*, is passed specifying the shortcut pathname.

Example

Listing 15.74 uses the `CreateShortcut()` method to create a Web site shortcut on the desktop to the `www.bestbookbuys.com` site.

Listing 15.74 *Creating a Desktop Shortcut with* `CreateShortcut()`

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Specify the Desktop as a folder.
myDesktop = WshShellObj.SpecialFolders("Desktop");

// Create a shortcut to the bestbookbuys.com website.
var urlShortcut = WshShellObj.CreateShortcut(myDesktop + "\\bestbookbuys.url");
urlShortcut.TargetPath = "http://www.bestbookbuys.com";
urlShortcut.Save();

</script>
</job>
```

WshShell.CurrentDirectory

WSH 5.6+

Syntax

`WshShell.CurrentDirectory`

Description

The `CurrentDirectory` property returns a string containing the full pathname of the current directory for the active process. This is not the directory that contains the WSH script, but the directory in which the process is begun.

Example

Listing 15.75 displays the current directory in which the example code is started from.

Listing 15.75 *Displaying the Current Directory with the* `CurrentDirectory` *Property*

```
<job>
<script language="JScript">
```

```
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Get current directory.
WScript.Echo(WshShellObj.CurrentDirectory);

</script>
</job>
```

WshShell.Environment

WSH 2.0+

Syntax

WshShell.Environment(str)

Description

The Environment property contains a WshEnvironment object, which is a collection of environment variables. An optional parameter, *str*, can be specified for the location of the environment variable. If *str* is not specified, the Environment property can return different types depending on the operating system. For Windows 95/98/Me, only "PROCESS" is permitted as the *str*. For Windows NT/2000, "System" can be specified.

Example

Listing 15.76 uses the Environment property to get the CLASSPATH environment variable and display it.

Listing 15.76 Using the Environment Property to Retrieve CLASSPATH

```
<job>
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create WshEnvironment object.
var WshEnv = WshShellObj.Environment("PROCESS");

// Get the classpath env var.
WScript.Echo(WshEnv("CLASSPATH"));

</script>
</job>
```

WshShell.Exec()

WSH 5.6+

Syntax

WshShell.Exec(cmd)

Description

The `Exec()` method of the `WshShell` object is used to run an application in a child command shell. The method takes a single parameter, *cmd*, specifying the command to start the application. This should be the same string as if you had typed it from the command line. A `WshScriptExec` object is returned, which provided access to the `StdIn`, `StdOut`, and `StdErr` streams.

Example

Listing 15.77 shows how the `Exec()` method is used to execute the `hello.wsf` script in a child command shell.

Listing 15.77 *Using the `Exec()` Method of `WshShell`*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// start hello.wsf in child command shell.
WshShellObj.Exec("cscript hello.wsf");

</script>
</job>
```

WshShell.ExpandEnvironmentStrings()

WSH 2.0+

Syntax

`WshShell.ExpandEnvironmentStrings(var)`

Description

The `ExpandEnvironmentStrings()` method is used to get an environment variable's expanded value. It takes a single parameter, *var*, which must be enclosed between % characters.

Example

Listing 15.78 shows an example of using the `ExpandEnvironmentStrings()` method to get the expanded version of the `PATH` environment variable.

Listing 15.78 *Using the `ExpandEnvironmentStrings()` Method*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");
```



```
// Echo expanded env var PATH.
WScript.Echo( WshShellObj.ExpandEnvironmentStrings("%path%") );

</script>
</job>
```

WshShell.LogEvent()

WSH 2.0+

Syntax

WshShell.LogEvent(eventType, logStr, [strTarget])

Description

The `LogEvent()` method is used to add an entry to a log file. The *eventType* and *logStr* must be specified. The *logStr* is a string value specifying the log entry text. Valid event types are listed in Table 15.11. There is also an optional parameter called *strTarget*, which is a string value indicating the name of the computer system where the event is stored. This optional parameter applies to Windows NT/2000 operating systems only.

Table 15.11 Event Types

Value	Description
0	SUCCESS
1	ERROR
2	WARNING
4	INFORMATION
8	AUDIT_SUCCESS
16	AUDIT_FAILURE

Example

Listing 15.79 shows how the `WshShell` object uses the `LogEvent()` method to log a successful event. When executed, the example logs a success event to the `WSH.log` file on Windows 9.x systems. If using Windows NT/2000, then messages are logged to the Windows NT event log file.

Listing 15.79 Logging an Event with `LogEvent()`

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Log a success event.
myFolder = WshShellObj.LogEvent(0, "Application successful");
```

Listing 15.79 Continued

```
</script>
</job>
```

WshShell.Popup()

WSH 2.0+

Syntax

```
WshShell.Popup(text, [wait], [title], [type])
```

Description

The `Popup()` method is used to display a piece of text in a pop-up message box. The `text` parameter must be specified for the text to display. The `wait` parameter is an optional parameter indicating the length of time (in seconds) for the pop-up to display. The `title` parameter is an optional string parameter specifying the title of the pop-up. The `type` parameter is an optional numeric parameter specifying the type of buttons and icons in the pop-up. Refer to Table 15.12 for available button types and Table 15.13 for available icon types. Values can be combined.

Table 15.12 Button Types for a Pop-up Message Box

Value	Description
0	Show OK button
1	Show OK and Cancel buttons
2	Show About, Retry, and Ignore buttons
3	Show Yes, No, and Cancel buttons
4	Show Yes and No buttons
5	Show Retry and Cancel buttons

Table 15.13 Icon Types for a Pop-up Message Box

Value	Description
16	Show "Stop Mark" Icon
32	Show "Question Mark" Icon
48	Show "Exclamation Mark" icon
64	Show "Information Mark" icon

Example

In Listing 15.80, a pop-up box is created with a sample message that displays for 15 seconds, has a title, and shows a specific combination of button and icon types.

Listing 15.80 Using the `Popup()` Method of the `WshShell` Object

```
<job>
<script language="JScript">
```

```
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create Popup.
WshShellObj.Popup("Sample message", 15, "My Popup box", 3+32);

</script>
</job>
```

WshShell.RegDelete()

WSH 2.0+

Syntax

WshShell.RegDelete(key)

Description

The `RegDelete()` method is used to delete a key or one of its values from the Windows registry. The *key* must be specified as the parameter. To indicate a key name, use a final backslash after the *key*. Do not use backslash to indicate a value. Abbreviated versions of root keys can be used. Table 15.14 lists these abbreviations.

Table 15.14 Root Key Abbreviations

Value	Description
HCKU	HKEY_CURRENT_USER
HKLM	HKEY_LOCAL_MACHINE
HKCR	HKEY_CLASSES_ROOT
HKEY_USERS	HKEY_USERS
HKEY_CURRENT_CONFIG	HKEY_CURRENT_CONFIG

CAUTION

Performing modifications to the Windows Registry can cause harm to the operation system if performed incorrectly. Use caution when working with the Windows Registry. It is suggested that a backup of the registry be made before deleting or writing to the registry.

Example

Listing 15.81 shows an example of how the `RegDelete()` method is used to delete a registry entry for Netscape.

Listing 15.81 Using the `RegDelete()` Method

```
<job>
<script language="JScript">
```

Listing 15.81 Continued

```
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Delete the Netscape key value.
WshShellObj.RegDelete("HKLM\\Software\\Netscape\\");

</script>
</job>
```

WshShell.RegRead()

WSH 2.0+

Syntax

WshShell.RegRead(*key*)

Description

The RegRead() method is used to read a key value or a value name from the registry. The *key* parameter is used to determine which key to read. To specify a key name, end the *key* with a final backslash. If backslash not specified, value name will be used.

Example

Listing 15.82 uses the RegRead() method to get the key value of the specified registry entry.

Listing 15.82 Using the RegRead() Method to Read a Registry Entry

```
<job>
<script language="JScript">
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Read the Netscape key value.
var value = WshShellObj.RegRead("HKLM\\Software\\Netscape\\
➤Netscape 6\\CurrentVersion");

WScript.Echo(value);

</script>
</job>
```

WshShell.RegWrite()

WSH 2.0+

Syntax

WshShell.RegWrite(*key*, *value*, [*type*])

Description

The `RegWrite()` method is used to create a new key, add another value name to an existing key, or modify the value of an existing value name. A parameter string, *key*, is used to specify the key name, value name, or value you want to add, create, or modify. The *value* parameter specifies the name of the new key to create, the name of the value to add, or the new value to assign to an existing key. The *type* parameter is an optional string type parameter indicating the value's data type. The available types are listed in Table 15.15.

Table 15.15 Root Key Abbreviations

Value	Description
REG_SZ	A string
REG_DWORD	A number
REG_BINARY	A binary value
REG_EXPAND_SZ	An expandable string

CAUTION

Performing modifications to the Windows Registry can cause harm to the operation system if performed incorrectly. Use caution when working with the Windows Registry. It is suggested that a backup of the registry be made before deleting or writing to the registry.

Example

Listing 15.83 shows how to write a new value to the registry.

Listing 15.83 Using the `RegWrite()` Method

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Write a new value.
WshShellObj.RegWrite("HKLM\\Software\\Netscape\\", 23, "REG_SZ");

</script>
</job>
```

WshShell.Run()

WSH 2.0+

Syntax

```
WshShell.Run(cmd, [windowStyle], [waitOnReturn])
```

Description

The `Run()` method runs a specified program in a new process. The program to be run is specified as the parameter `cmd`. Two optional parameters can also be specified. The `windowStyle` parameter is an integer indicating the appearance of the program's window. The valid window styles are listed in Table 15.16. The `waitOnReturn` parameter is a boolean value indicating whether the script should wait before executing the next script command.

Table 15.16 *Valid Windows Styles*

Style	Description
0	Hides window and activates another window
1	Activates and displays a window
2	Activates the window and displays it as minimized
3	Activates the window and displays it as maximized
4	Displays a window in the most recent size and position
5	Activates the window and displays it in its current size and position
6	Minimizes the window and activates the next top level window
7	Displays the window as minimized
8	Displays the window in its current state
9	Restores a window to its original size and position
10	Sets the show state based on the program that started the application

Example

In Listing 15.84 the `Run()` method is used to startup the Notepad application in the minimized state.

Listing 15.84 *Using the `Run()` method of the `WshShell` object*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Start up Notepad in the minimize state.
WshShellObj.Run("notepad",7)

</script>
</job>
```

WshShell.SendKeys()

WSH 2.0+

Syntax

```
WshShell.SendKeys(keyStr)
```

Description

The `SendKeys()` method is used to send one or more keystrokes to the active window. The keys to be sent are specified as the *keyStr* parameter. This method is typically used for applications that have no user interface. If sending a combination of keystrokes, simply group them together in a string. For instance, if sending the keys x, y, and z, use `xyz`.

There are some special keys that require a certain syntax. For the +, ^, %, ~, [, and], enclose those symbols in braces ({}). Other keystrokes that do not generate any characters have special syntax's as well. They are listed in Table 15.17.

Table 15.17 *Special Keystrokes*

Keystroke	Syntax
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER}
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}

Table 15.17 *Continued*

Keystroke	Syntax
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

Example

In Listing 15.85 a WshShell object is created. Then the Run() method is called to start the Notepad application. Once the Notepad application is up, it is activated using the AppActivate() method. After being activated, a sequence of keys are sent to the application.

Listing 15.85 *Sending Keystrokes to Notepad with* SendKeys()

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Start the Notepad application.
WshShellObj.Run("Notepad");

// Activate the Notepad application.
WshShellObj.AppActivate("Notepad");

// Send keys to the notepad app.
WshShellObj.SendKeys("I love Pure JavaScript!");

</script>
</job>
```

WshShell.SpecialFolders

WSH 2.0+

Syntax

```
WshShell.SpecialFolders(folder )
```

Description

The SpecialFolders property returns a collection of special folders. Special folders within the Windows operating system refer to predefined folders that are built into the OS. Some examples are the “Desktop”, “Favorites”, and “Startup” folder.

Example

In Listing 15.86 a WshShell object is first created. Then the SpecialFolders property is used to create a WshSpecialFolders object.

Listing 15.86 Using the SpecialFolders Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a Special Folders Object.
myFolder = WshShellObj.SpecialFolders("Desktop");

</script>
</job>
```

WshShell.WshShortcut

WSH 2.0+

Syntax

WshShell.WshShortcut

Description

The WshShortcut object is a child object of WshShell. Because it is not directly accessible through the *WshShell.WshShortcut* syntax, see the entry for WshShortcut in this chapter for more information.

WshShell.WshUrlShortcut

WSH 2.0+

Syntax

WshShell.WshUrlShortcut

Description

The WshUrlShortcut object is a child object of WshShell. Because it is not directly accessible through the *WshShell.WshUrlShortcut* syntax, see the entry for WshUrlShortcut in this chapter for more information.

WshShell.WshEnvironment

WSH 2.0+

Syntax

WshShell.WshEnvironment

Description

The `WshEnvironment` object is a child object of `WshShell`. Because it is not directly accessible through the `WshShell.WshEnvironment` syntax, see the entry for `WshEnvironment` in this chapter for more information.

WshShell.WshSpecialFolders

WSH 2.0+

Syntax

`WshShell.WshSpecialFolders`

Description

The `WshSpecialFolders` object is a child object of `WshShell`. Because it is not directly accessible through the `WshShell.WshSpecialFolders` syntax, see the entry for `WshSpecialFolders` in this chapter for more information.

WshShell.WshScriptExec

WSH 5.6+

Syntax

`WshShell.WshScriptExec`

Description

The `WshScriptExec` object is a child object of `WshShell`. Because it is not directly accessible through the `WshShell.WshScriptExec` syntax, see the entry for `WshScriptExec` in this chapter for more information.

WshShortcut

WSH 2.0+

Syntax

`WshShortcut`

Description

The `WshShortcut` object is used to create an object reference to a shortcut. It is instantiated by using the `createShortcut` method of the `WshShell` object. Upon being instantiated, it is stored in memory and not written to disk until the `Save` method is called. Table 15.18 lists all the properties and methods associated with the `WshShortcut` object.

Table 15.18 Properties and Method Associated with the WshShortcut Object

Type	Item	Description
Property	Arguments	Returns a pointer to the WshArguments collection
	Description	Describes the shortcut
	FullName	Returns the full path to the host executable or shortcut
	Hotkey	Provides a hotkey to the shortcut
	IconLocation	Provides the icon location of the shortcut
	TargetPath	Sets a path to the shortcut executable file
	WindowStyle	Assigns a Windows style to the shortcut
	WorkingDirectory	Assigns or identifies a working directory for the shortcut
Method	Save()	Saves shortcut to disk

Example

Listing 15.87 shows how the WshShortcut object is instantiated within memory, assigned a target path, and then saved to disk.

Listing 15.87 Creating a WshShortcut Object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\gopack.url");
urlShortcut.TargetPath = "http://www.gopack.com";
urlShortcut.Save();

</script>
</job>
```

WshShortcut.Arguments

WSH 2.0+

Syntax

WshShortcut.Arguments

Description

The Arguments property provides access to the arguments collection through the WshArguments object.

Example

In Listing 15.88 creates a new `WshShortcut` object and then sets the path to the argument value entered. So if the script named `listing88.wsf` were run from the command line with the following command:

```
cscript listing88.wsf http://www.w3c.org
```

The shortcut would be created with the target pointing to the entered URL. Then the target path is output to the user.

Listing 15.88 *Using the Arguments Property of WshShortcut*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a WshArguments object with the Arguments property.
var myArgs = WScript.Arguments;

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\myShortcut.url");
urlShortcut.TargetPath = myArgs.item(0);

WScript.Echo("The target path is: " + urlShortcut.TargetPath);

</script>
</job>
```

WshShortcut.Description

WSH 2.0+

Syntax

`WshShortcut.Description`

Description

The `Description` property returns a string value containing the description of the shortcut.

Example

In Listing 15.89 a shortcut object is created and the description is set. When executed, the description is displayed to the user.

Listing 15.89 *Using the Description Property*

```
<job>
<script language="JScript">
```

```
// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");

// Set the Description property.
myShortcut.Description = "NC State Wolfpack";

WScript.Echo("The shortcut description is: " + myShortcut.Description);

</script>
</job>
```

WshShortcut.FullName

WSH 2.0+

Syntax

WshShortcut.FullName

Description

The `FullName` property of the `WshShortcut` object returns a string containing the fully-qualified path to the shortcut. It is a read-only property.

Example

In Listing 15.90 a shortcut is created and the fully-qualified path is displayed using the `FullName` property.

Listing 15.90 *Using the `FullName` Property of the `WshShortcut` Object*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");

// Echo the FullName of the shortcut.
WScript.Echo("The full path is: " + myShortcut.FullName);

</script>
</job>
```

WshShortcut.Hotkey

WSH 2.0+

Syntax

WshShortcut.Hotkey = *hotKeyStr*

Description

The *HotKey* property is used to assign a key combination to a shortcut or to identify the key combination assigned to a shortcut. The *hotKeyStr* parameter is the key combination to be assigned. It is in the form of *[keyModifier]KeyName*. The valid *keyModifiers* are Alt+, Ctrl+, Shift+, Ext+. (Ext+ means an “Extended Key”, which refers to any new type of shift key added in the future.) Valid *KeyName* identifiers are all alphanumeric keys and F1–F12 function keys.

Example

Listing 15.91 shows how to create a shortcut and then assign a hotkey to the shortcut.

Listing 15.91 *Assigning a Hotkey to a WshShortcut Object*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");
myShortcut.HotKey = "Alt+Shift+p";

// Echo the shortcut's Hotkey.
WScript.Echo("The hotkey is: " + myShortcut.Hotkey);

</script>
</job>
```

WshShortcut.IconLocation

WSH 2.0+

Syntax

WshShortcut.IconLocation

WshShortcut.IconLocation = *iconLocationStr*

Description

The *IconLocation* property assigns an icon to a shortcut or identifies the icon associated with a shortcut. If used to assign an icon, the parameter *iconLocationStr* must be used. It should include a string to the fully-qualified path to the icon file and an index that identifies the icon in the collection of icons contained in the file.

Example

Listing 15.92 shows a shortcut which is created and assigned an icon.

Listing 15.92 *Using the IconLocation Property*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");
myShortcut.IconLocation = "iexplorer.exe,0";

// Echo the shortcut's IconLocation.
WScript.Echo("The icon location is: " + myShortcut.IconLocation);

</script>
</job>
```

WshShortcut.Save()

WSH 2.0+

Syntax

WshShortcut.Save()

Description

The Save() method of the WshShortcut object is used to save the shortcut to disk. Without using the Save() method, the shortcut is only stored in memory.

Example

Listing 15.93 shows a shortcut to the gopack.com site being created and then saved to disk with the Save() method.

Listing 15.93 *Saving a shortcut to disk with the Save() method*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\gopack.url");
urlShortcut.TargetPath = "http://www.gopack.com";
```

Listing 15.93 Continued

```
// Save the shortcut to disk.
urlShortcut.Save();

</script>
</job>
```

WshShortcut.TargetPath

WSH 2.0+

Syntax

WshShortcut.TargetPath

Description

The TargetPath property is used to assign a path to the executable file to which the shortcut points. It can also be used to identify the path to the shortcut executable file.

Example

Listing 15.94 uses the TargetPath property to set the target URL for the urlShortcut object. Then the path is displayed to the user using the same TargetPath property.

Listing 15.94 Using the TargetPath Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\gopack.url");

// Set the TargetPath property.
urlShortcut.TargetPath = "http://www.gopack.com";

// Echo the TargetPath of the urlShortcut.
WScript.Echo("The target path is: " + urlShortcut.TargetPath);

</script>
</job>
```

WshShortcut.WindowStyle

WSH 2.0+

Syntax

WshShortcut.WindowState

WshShortcut.WindowState = *styleValue*

Description

The WindowStyle property is used to identify a window style of a shortcut. It can also be used to assign a window style with a shortcut. If assigning a style, then a *styleValue* must be specified. Valid styles are listed in Table 15.19.

Table 15.19 Window Styles

Value	Description
1	Activates and displays a window
3	Activates the window and displays it as a maximized window
7	Minimizes the window and activates the next top-level window

Example

Listing 15.95 creates a new shortcut and assigns a window style to it.

Listing 15.95 Using the WindowStyle Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");

// Set the window style.
myShortcut.WindowStyle= 3;

</script>
</job>
```

WshShortcut.WorkingDirectory

WSH 2.0+

Syntax

WshShortcut.WorkingDirectory

Description

The WorkingDirectory property is used to identify a working directory of a shortcut. It can also be used to assign a working directory to a shortcut.

Example

Listing 15.96 creates a shortcut and then assigns the `WorkingDirectory` property with a specified working directory.

Listing 15.96 Using the `WorkingDirectory` Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var myShortcut = WshShellObj.CreateShortcut("c:\\gopack.lnk");

// Set the WorkingDirectory property.
myShortcut.WorkingDirectory = "c:\\My Documents";

// Echo the working directory value.
WScript.Echo("The working directory is: " + myShortcut.WorkingDirectory);

</script>
</job>
```

WshSpecialFolders

WSH 2.0+

Syntax

WshSpecialFolders

Description

The `WshSpecialFolders` object returns a collection of special folders. This collection is used to retrieve paths to special folders such as Desktop, Start Menu, and My Documents. The different types of special folders and their supported operating system are listed in Table 15.20. In Table 15.21, all the properties and methods associated with the `WshSpecialFolders` object are listed.

Table 15.20 Special Folder Types

Folder Name	Windows Version
AllUsersDesktop	Windows 2000
AllUsersStartMenu	Windows 2000
AllUsersPrograms	Windows 2000
AllUsersStartup	Windows 2000
Desktop	Windows 95/98, Windows 2000
Favorites	Windows 95/98, Windows 2000

Folder Name	Windows Version
Fonts	Windows 95/98, Windows 2000
My Documents	Windows 95/98, Windows 2000
NetHood	Windows 95/98, Windows 2000
PrintHood	Windows 95/98, Windows 2000
Programs	Windows 95/98, Windows 2000
Recent	Windows 95/98, Windows 2000
SendTo	Windows 95/98, Windows 2000
Start Menu	Windows 95/98, Windows 2000
StartupB	Windows 2000
Templates	Windows 2000

Table 15.21 *Properties and Methods Associated with WshSpecialFolders Object*

Type	Item	Description
Property	Item	Retrieves a specific item in the collection
	length	Returns the number of enumerated items (Same as the Count() method)
Method	Count()	Returns the number of enumerated items (Same as the length property)

Example

The code in Listing 15.97 creates a new WshSpecialFolders object referencing the Desktop special folder.

Listing 15.97 Creating a WshSpecialFolder Object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a Special Folders Object.
myFolder = WshShellObj.SpecialFolders("Desktop");

</script>
</job>
```

WshSpecialFolders.Count

WSH 2.0+

Syntax

WshSpecialFolders.Count

Description

The Count method returns the number of enumerated items in the WshSpecialFolders collection. This method provides the same functionality as the length property.

Example

Listing 15.98 creates a new WshSpecialFolders object and then displays the count of the special folders collection.

Listing 15.98 Using the Count Method of the WshSpecialFolders Object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a Special Folders Object.
WshSpecialFolders = WshShellObj.SpecialFolders("Desktop");

// Echo the count of special folders using the Count method.
WScript.Echo("The special folder Count is: " + WshSpecialFolders.Count );
</script>
</job>
```

WshSpecialFolders.Item

WSH 2.0+

Syntax

WshSpecialFolders.Item(index)

Description

The Item property exposes a specific item from the WshSpecialFolders collection. The item to be exposed is specified in the *index* parameter.

Example

Listing 15.99 uses the Item property to loop through the special folders collection and get the items.

Listing 15.99 Using the Item Property of the WshSpecialFolders object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a Special Folders Object.
WshSpecialFolders = WshShellObj.SpecialFolders("Desktop");
```

```
for(i=0; i< WshSpecialFolders.length; i++)
{
    // Echo each item in the special folders collection.
    WScript.Echo(i + " = " + WshSpecialFolders.Item(i) );
}

</script>
</job>
```

WshSpecialFolders.length

WSH 2.0+

Syntax

WshSpecialFolders.length

Description

The `length` property returns the number of enumerated items in the `WshSpecialFolders` collection. This property provides the same functionality as the `Count()` method.

Example

Listing 15.100 shows how the `length` property is used to display the number of enumerated items in the `WshSpecialFolders` object.

Listing 15.100 Displaying the length of the `WshSpecialFolders` collection

```
<job>
<script language="JScript">

    // Create a WshShell object.
    WshShellObj = WScript.CreateObject("WScript.Shell");

    // Create a Special Folders Object.
    WshSpecialFolders = WshShellObj.SpecialFolders("Desktop");

    // Echo the number of items in the special folder collection using length.
    WScript.Echo("The special folder length is: " + WshSpecialFolders.length );
</script>
</job>
```

WshUnnamed

WSH 5.6+

Syntax

WshUnnamed

Description

The WshUnnamed object provides access to the read-only collection of arguments without associated names. The WshUnnamed object is returned from the Unnamed property of the WshArguments object. A named parameter is one that is made up of a key/value pair in which the key and value are separated with a colon (:) and the key is preceded by a forward slash (/). Table 15.22 lists all the properties and methods associated with the WshUnnamed object.

Table 15.22 Properties and Method Associated with the WshUnnamed Object

Type	Item	Description
Property	Item	Provides access to items in the WshUnnamed collection
	length	Returns the number of command-line parameters
Method	Count()	Returns the number of switches in the WshUnnamed object

Example

Listing 15.101 creates a WshArgument object, called *WshArgsObj*. The Unnamed property is then used to access the WshUnnamed object, which is used to display the unnamed parameters, if such parameters should exist. For example, let us assume that the script named *listing101.wsf* was run using the following line:

```
cscript listing101.wsf /color:green blue red
```

The string and then the unnamed parameters *blue* and *red* would be displayed.

Listing 15.101 Displaying all the Unnamed Objects

```
<job>
<script language="JScript">

// Get collection of arguments.
WshArgsObj = WScript.Arguments;

// Get collection of unnamed arguments.
WshUnnamedArgsObj = WshArgsObj.Unnamed

if(WshUnnamedArgsObj.length)
{
    WScript.Echo("Unnamed Arguments:");

    // Loop through all the arguments in unnamedcollection.
    for(i=0; i< WshUnnamedArgsObj.Length; i++)
    {
        WScript.Echo(i + " = " + WshUnnamedArgsObj.Item(i));
    }
}
else
```

```
{  
    WScript.Echo("No unnamed arguments");  
}  
  
</script>  
</job>
```

WshUnnamed.Count()

WSH 5.6+

Syntax

WshUnnamed.Count()

Description

The Count() method returns the number of WshUnnamed objects in the collection.

Example

Listing 15.102 creates a WshArgument object, called *WshArgsObj*. The Unnamed property is then used to access the WshUnnamed object, which is used to display the count of unnamed parameters, if such parameters should exist. For example, let us assume that the script named listing102.wsf was run using the following line:

```
cscript listing102.wsf /color:green blue red yellow
```

The total number of unnamed parameters, 3, would be displayed.

Listing 15.102 Getting the Count of the Unnamed Parameters with Count()

```
<job>  
<script language="JScript">  
  
    // Get collection of arguments.  
    WshArgsObj = WScript.Arguments;  
  
    // Get collection of unnamed arguments.  
    WshUnnamedArgsObj = WshArgsObj.Unnamed  
  
    if(WshUnnamedArgsObj.length)  
    {  
        // Echo the count of unnamed parameters.  
        WScript.Echo("The count of WshUnnamed object is: " + WshUnnamedArgsObj.Count);  
    }  
    else  
    {  
        WScript.Echo("No unnamed arguments");  
    }  
}
```

Listing 15.102 Continued

```
</script>
</job>
```

WshUnnamed.Item

WSH 5.6+

Syntax

WshUnnamed.Item

Description

The *Item* property provides access to each item in the *WshUnnamed* collection.

Example

Listing 15.103 creates a *WshArgument* object, called *WshArgsObj*. The *Unnamed* property is then used to access the *WshUnnamed* object, which is used to display the count of unnamed parameters, should such parameters exist. For example, let us assume that the script named *listing103.wsf* was run using the following line:

```
cscript listing103.wsf /color:green blue red yellow orange
```

Then each *WshUnnamed* item would be displayed.

Listing 15.103 Displaying Each Unnamed Item

```
<job>
<script language="JScript">

// Get collection of arguments.
WshArgsObj = WScript.Arguments;

// Get collection of unnamed arguments.
WshUnnamedArgsObj = WshArgsObj.Unnamed

if(WshUnnamedArgsObj.length)
{
    WScript.Echo("Unnamed Arguments:");

    // Loop through all the arguments in unnamedcollection.
    for(i=0; i< WshUnnamedArgsObj.length; i++)
    {
        WScript.Echo(i + " = " + WshUnnamedArgsObj.Item(i));
    }
}

</script>
</job>
```


WshUnnamed.length

WSH 5.6+

Syntax

WshUnnamed.length

Description

The length property returns the number of command-line arguments passed to a script. This is the same as the Count property.

Example

The code in Listing 15.104 creates a WshArgument object, called *WshArgsObj*. The Unnamed property is then used to access the WshUnnamed object, which is used to display the count of unnamed parameters, if such parameters should exist. For example, let us assume that the script named listing104.wsf was run using the following line:

```
cscript listing104.wsf /color:green blue red yellow
```

Then the WshUnnamed length, 3, would be displayed.

Listing 15.104 Using the length *property*

```
<job>
<script language="JScript">

    // Get collection of arguments.
    WshArgsObj = WScript.Arguments;

    // Get collection of unnamed arguments.
    WshUnnamedArgsObj = WshArgsObj.Unnamed

    // Check the length of the WshUnnamed object.
    if(WshUnnamedArgsObj.length)
    {
        // Use the length property to echo the number of line arguments.
        WScript.Echo("The WshUnnamed length is: " + WshUnnamedArgsObj.length);
    }
    else
    {
        WScript.Echo("No unnamed arguments");
    }
</script>
</job>
```

WshUrlShortcut

WSH 2.0+

Syntax

WshUrlShortcut

Description

The WshUrlShortcut object creates a shortcut to an Internet resource. It is instantiated by using the createShortcut method of the WshShell object. Upon being instantiated, it is stored in memory and not written to disk until the Save method is called. Table 15.23 lists all the properties and methods associated with the WshUrlShortcut object.

Table 15.23 *Properties and Method Associated with the WshUrlShortcut Object*

Type	Item	Description
Property	FullName	Returns fully qualified path to shortcut
	TargetPath	Assigns or identifies path to shortcut executable
Method	Save()	Writes a shortcut to disk

Example

Listing 15.105 shows how to create a new WshUrlShortcut object. An Internet link to the www.borders.com site is created within memory.

Listing 15.105 *Creating a New WshUrlShortcut Object*

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

//Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\\\borders.url");
urlShortcut.TargetPath = "http://www.borders.com";

</script>
</job>
```

WshUrlShortcut.FullName

WSH 2.0+

Syntax

WshUrlShortcut.FullName

Description

The FullName property is a read-only string that contains the fully-qualified path to the WshUrlShortcut object.

Example

Listing 15.106 uses the `FullName` property to display its value.

Listing 15.106 Using the `FullName` Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\\\borders.url");
urlShortcut.TargetPath = "http://www.borders.com";

// Echo the FullName of the WshUrlShortcut object.
WScript.Echo("The FullName is: " + urlShortcut.FullName);

</script>
</job>
```

WshUrlShortcut.Save()

WSH 2.0+

Syntax

```
WshUrlShortcut.Save()
```

Description

The `Save()` method is used to save the `WshUrlShortcut` object to disk. When the object is initially created, it lives only in memory and is not written to disk unless the `Save()` method is used.

Example

Listing 15.107 uses the `Save()` method to store the shortcut to disk.

Listing 15.107 Using the `Save()` Method of the `WshUrlShortcut` Object

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\\\bmw.url");
urlShortcut.TargetPath = "http://www.bmwusa.com";
```

Listing 15.107 Continued

```
// Save the shortcut to disk.
urlShortcut.Save();

</script>
</job>
```

WshUrlShortcut.TargetPath

WSH 2.0+

Syntax

WshUrlShortcut.TargetPath

Description

The TargetPath property is used to either assign or identify the path to the executable file for which the shortcut was created.

Example

In Listing 15.108 a WshUrlShortcut object is created and its TargetPath property is set.

Listing 15.108 Using the TargetPath Property

```
<job>
<script language="JScript">

// Create a WshShell object.
WshShellObj = WScript.CreateObject("WScript.Shell");

// Create a shortcut.
var urlShortcut = WshShellObj.CreateShortcut("c:\\\\vatech.url");
urlShortcut.TargetPath = "http://www.vt.edu";

// Display the TargetPath content.
WScript.Echo("The TargetPath is: " + urlShortcut.TargetPath);

</script>
</job>
```




INDEX

Special Note to Readers

Chapters 1-9 are located in the book. Chapters 10-15 appear on the CD. The indexes found in both the book and the CD are identical.

When you look up information in either index, you will notice that some page numbers have CD: before them. These refer to pages found only on the CD. For example, the following listing lets you know whether information is located in the book or on the CD:

checked property, 615, CD:1585, CD:1590

Symbols

~ (bitwise NOT operator), 224
: (colon), 72-74, 284, 573, 580, 589, CD:2016
, (comma), 214-215
. (dot) (pattern matching character), 86
(hash symbol), 578, 885
" " (double quotation marks), 35, 1215
' ' (single quotation marks), 35
_ (underscore), 37
() (parentheses), 61
(...) (pattern matching character), 87
\$ (pattern matching character), 87
\$1,\$2,...\$9 property, 474
-- (decrement operator), 48-49, 204-205
--> comment tags (HTML), 33
{ } (curly brackets), 63-65, 77
{x} (pattern matching character), 87
{x,y} (pattern matching character), 86
! (logical NOT operator), 55, 205-206
!= (not equal operator), 57, 206-207

!= (non-identity operator), 58, 207-208
 * (pattern matching character), 87
 * (multiplication operator), 47, 213
 *= (multiplication assignment operator), 52, 213-214
 = (assignment operator), 37, 232
 == (equal operator), 56-57, 233-234
 === (identity operator), 58, 234-235
 ? (question mark), 573, 886, 1323
 ? (pattern matching character), 87
 ?: (conditional operator), 58-59, 218-219
 & (bitwise AND operator), 60, 210-211
 &= (bitwise AND assignment operator), 53, 212-213
 && (logical AND operator), 211-212
 ^ (bitwise exclusive OR operator), 219
 ^ (pattern matching character), 87
 ^ (XOR bitwise operator), 60
 ^= (bitwise exclusive OR assignment operator), 53, 220-221
 | (pattern matching character), 87
 | (OR bitwise operator), 60, 221, 646
 |= (bitwise OR assignment operator), 53, 223-224
 || (logical OR operator), 55, 222-223
 % (modulus operator), 47, 208-209
 %= (modulus assignment operator), 209-210
 %== (assignment) operator, 53
 %% (AND) logical operator, 55
 - (bitwise NOT operator), 60
 - (subtraction operator), 46-47, 202-203
 - (unary negation operator), 49, 203-204
 -= (subtraction assignment operator), 52, 232-233
 + (addition operator), 46, 49-50, 53, 225
 + (pattern matching character), 87
 + (plus sign), 563
 += (addition assignment operator), 52, 227-228
 ++ (increment operator), 48, 226-227
 [] (square brackets) operator, 40, 250
 [...] (pattern matching character), 86
 [b] (pattern matching character), 86
 [^...] (pattern matching character), 86
 [^x-y] (pattern matching character), 86
 [x-y] (pattern matching character), 86
 < (less than comparison operator), 57-58, 228-229
 <= (less than or equal operator), 58, 231-232
 <!-- comment tags (HTML), 33

<!-- --> comment tags (HTML), 33
 << (shift left operator), 61, 229-230
 <<= (shift left assignment operator), 53, 230-231
 > (greater than operator), 57, 235-236
 >= (greater than or equal operator), 58, 236-237
 >> (shift right with sign operator), 61, 237-238
 >>= (shift right with sign assignment operator), 53, 238-239
 >>> (shift right zero fill operator), 61, 239-240
 >>>= (shift right zero fill assignment operator), 53, 240-241
 @cc_on statement, 200
 @if statement, 200-201
 @set property, 201-202
 @_alpha variable, 193-194
 @_jscript build variable, 195
 @_jscript variable, 194
 @_jscript version variable, 196
 @_jscript_build variable, 195
 @_jscript_version variable, 195
 @_mac variable, 196
 @_mc680x0 variable, 197
 @_PowerPC variable, 197-198
 @_win16 variable, 198
 @_win32 variable, 199
 @_x86 variable, 199-200
 / (division operator), 47, 215
 / (forward slash), 86, 573, 587, CD:2016
 /n character, 1290
 /r character, 1290
 /= (division assignment operator), 52, 217
 /* */ comment tags (C), 33
 /* / (multiline comments), 216
 // (single-line comments), 33, 216-217
 //? (wscript.exe and cscript.exe option), 185
 //B (wscript.exe and cscript.exe option), 185
 //D (wscript.exe and cscript.exe option), 185
 //E:engine (wscript.exe and cscript.exe option), 185
 //H:cscript (wscript.exe and cscript.exe option), 185
 //H:wscript (wscript.exe and cscript.exe option), 185
 //I (wscript.exe and cscript.exe option), 185
 //Job:<jobID> (wscript.exe and cscript.exe option), 185

//logo (wscript.exe and cscript.exe option), 185
 //nologo (wscript.exe and cscript.exe option), 185
 //S (wscript.exe and cscript.exe option), 185
 //T:nn (wscript.exe and cscript.exe option), 185
 //U (wscript.exe and cscript.exe option), 185
 //X (wscript.exe and cscript.exe option), 185
 \ (backslash), 35, 87
 \' (escape sequence), 36
 \" (escape sequence), 35-36
 \ (literal character), 88
 \) (literal character), 88
 \[(literal character), 88
 \] (literal character), 88
 \{ (literal character), 88
 \} (literal character), 88
 \| (literal character), 87
 * (literal character), 87
 \+ (literal character), 87
 \. (literal character), 87
 \V (literal character), 87
 \? (literal character), 87
 \B (pattern matching character), 87
 \cX (literal character), 88
 \D (pattern matching character), 86
 \f (literal character), 87
 \inetpub\ASP\Samp\Samples directory, 1320
 \n (literal character), 87
 \r (literal character), 87
 \s (pattern matching character), 86
 \t (literal character), 87
 \v (literal character), 87
 \W (pattern matching character), 86
 \x (pattern matching character), 87
 \xHH (literal character), 88
 \XXX (literal character), 88
 \\ escape sequence, 36
 \\ (literal character), 87

A

A drive readiness, code to determine, CD:1838
 a option (open() method), 1275, 1286
 <a> tag, formatting, code, 1192
 -a version command-line parameter, 160
 Abandon() method (Session object), 1379-1380

abbr property (HTMLTableCellElement object), CD:1674-1675
 abbreviations, root key, CD:1997-1999
 Abort event, 90
 ABORT
 event (Event object), 698
 property, code, 720-721
 above property (Layer object), 821-824
 abs() method (Math object), 408-410
 abstract keyword, 241
 accept property (HTMLInputElement object), CD:1585-1587
 acceptCharset property (HTMLFormElement object), CD:1546-1548
 accessKey property
 code, CD:1481-1482, CD:1514, CD:1587-1588, CD:1603
 HTMLAnchorElement object, CD:1480
 HTMLAreaElement object, CD:1497
 HTMLButtonElement object, CD:1513
 HTMLInputElement object, CD:1585
 HTMLLabelElement object, CD:1602
 HTMLLegendElement object, CD:1605
 HTMLTextAreaElement object, CD:1715
 acos() method (Math object), 408-411
 action property
 code, CD:1548
 Form object, 761
 HTMLFormElement object, CD:1546
 actions of forms, 656-657
 Active Server Pages (ASPs), 24-25, 168-169 177
 ActiveXObject
 method, CD:1825
 object, 241-242
 ad tags, 18
 add() method, code, CD:1659-1661, CD:1909
 Add() method (Dictionary object), CD:1825-1826
 Add property (Folders collection), CD:1909
 Add Windows PrinterConnection() method (WshNetwork object), CD:1964
 addClient() function, 1176, 1316
 addEventListener() method (EventTarget object), CD:1744-1746
 AddHeader() method (Response object), 1335

addition assignment operator (+=), 227-228

ADDITION constant (MutationEvent object), CD:1756

addition operator (+), 46, 49-50, 53, 225

AddPrinterConnection() method (WshNetwork object), CD:1964-1965

addResponseHeader() function, 1176-1177

AddWindowsPrinterConnection() method, code, CD:1966

advanced assignment, 52-53

agent property (request object), 1327-1328

alert boxes

code, 1090

displaying, code, 606-609, 627-628

onResize event handler code, 1133

type properties, code to display, 992, 1041-1042, 1059-1060, 1080-1081

alert method (Window object), 1085

alert() method, 103-106, 1090

align parameter (blobImage() method), 1190

align property

code, 998, CD:1491-1493, CD:1523, CD:1564-1566

HTMLAppletElement object, CD:1490

HTMLDivElement object, CD:1523

HTMLHeadingElement object, CD:1564

HTMLHRElement object, CD:1565

HTMLIFrameElement object, CD:1570

HTMLImageElement object, CD:1577

HTMLInputElement object, CD:1585

HTMLLegendElement object, CD:1605

HTMLObjectElement object, CD:1624

HTMLParagraphElement object, CD:1648

HTMLTableCaptionElement object, CD:1673

HTMLTableCellElement object, CD:1674

HTMLTableColElement object, CD:1684

HTMLTableElement object, CD:1688

HTMLTableRowElement object, CD:1703

HTMLTableSectionElement object, CD:1710

Style object, 649, 662, 688, 995

aLink property (HTMLBodyElement object), CD:1507-1509

alinkColor property, 636-638, 1108

ALL HTTP variable (ServerVariables collection), 1324

all property (Document object), 636

ALL RAW variable (ServerVariables collection), 1324

ALT MASK modifier, code to access, 707

alt property

code, CD:1491, CD:1499, CD:1579, CD:1589

HTMLAppletElement object, CD:1490

HTMLAreaElement object, CD:1497

HTMLImageElement object, CD:1577

HTMLInputElement object, CD:1585

altKey property (MouseEvent object),

CD:1748-1749

altText parameter (blobImage() method), 1190

alwaysLowered parameter (Window object), 1134

alwaysRaised parameter (Window object), 1134

anchor length property, hyperlinks, 642-643

anchor() method (String object), 498

Anchor object, 564-565

name property, 566-567

text property, 567-568

unwatch() method, 568-569

watch() method, 569-570

x property, 570-571

y property, 571

anchors, iterating, code, CD:1526-1527

anchors property

array, anchor names, 642

Document object, 636

HTMLDocument object, CD:1525

window.document property, 1108

AND (%) logical operator, 55

AND (&) bitwise operator, 60

AOL, Web browser, 108

AppActivate() method (WshShell object), CD:1990-1991

appCodeName property (navigator object), 904-905

appendChild() method (Node object), CD:1455-1457

appendData() method (CharacterData object), CD:1407, CD:1410

AppendToLog() method (Response object), 1335-1336

APPL MD PATH variable

(ServerVariables collection), 1324

APPL PHYSICAL PATH variable (ServerVariables collection), 1324

Apple Macintosh system, code to alert when on, 196

applet array, length property, 116-117

Applet object, 572

applets (Java)

accessing, code, 644-645

applet array, 116-118

call() method, code, 398-399

checking, code, 116-117

iterating, code, CD:1527-1528

JavaScript, code, 401

loading, HTML code, 122-123

text, changing, 122-123

applets property

array, code to access applets, 644

Document object, 636

HTMLDocument object, CD:1525

window.document property, 1108

Application object, 168, 1177-1178

Application.OnEnd event (Application object), 1178

Application.OnStart event (Application object), 1178

Application.Application_OnEnd event function, 1178

Application.Application_OnStart event function, 1179

Application.Contents collection, 1179-1180

Application.Contents.Lock() method, 1181-1182

Application.Contents.Remove() method, 1180

Application.Contents.RemoveAll() method, 1180-1181

Application.StaticObjects collection, 1182
Application.Unlock method, 1182-1183
applications

binaries, COM servers, CD:1824

Debugging window, 166

e-mail pages, building, 174

jsac (JavaScript Application Compiler), 160

locking, code, 1305

Microsoft, JScript support, CD:1823

Server Side JavaScript Application Manager, 161-167

settings (default), for Server Side JavaScript Application Manager, 164-165

Web-based, 27

apply() method (Function object), 353-354

appName property (navigator object), 904-906

appVersion property (navigator object), 904-907

architectures, WSH (Windows Script Host), 180-181

archive property

code, CD:1492, CD:1626

HTMLAppletElement object, CD:1490

HTMLObjectElement object, CD:1624

Area object

Areas, code to create, 574-576

description, 573

event handlers, 574

events, code to pass, 577-578

handleEvent() method, 573

hash property, 573, 579-580

host property, 573, 581

hostname property, 573, 582-583

href property, 573, 583-584

methods, 573

onDbClick event handler, 574, 585

onMouseOut event handler, 574, 585-586

onMouseOver event handler, 574, 586-587

pathname property, 573, 587-588

port property, 573, 588-589

properties, 573

protocol property, 573, 589-590

search property, 573, 590-591

syntax, 573

target property, 573, 592-593

unwatch() method, 573, 594-595

watch() method, 573, 596-597

Area.handleEvent() method, 576-577

Area.hash property, 578

Area.host property, 580

Area.hostname property, 582

Area.href property, 583

Area.onDbClick event handler, 584-585

Area.onMouseOut event handler, 585

Area.onMouseOver event handler, 586

Area.pathname property, 587

Area.port property, 588

Area.protocol property, 589

Area.search property, 590

Area.target property, 591

Area.unwatch() method, 594

Area.watch() method, 596

areas

- creating, code, 574-576
- properties, code to access, 574-576

areas property, CD:1615-1616**arg1,:argN argument**

- Array object, 269
- concat() method, 251
- push() method, 259

arg3,:argN argument (Array object), 266**arg11,:argN argument (Array object), 249****arguments**

- ActiveXObject object, 242
- arg1,:argN, 251, 259, 269
- arg3,:argN (Array object), 266
- arg11,:argN (Array object), 249
- arguments array, 79
- Array object, 249
- arrays, code to sort, 264-265
- Attr object, CD:1404
- Boolean object, 273
- borderWidths() method, 1005
- Button object, 597
- CharacterData object, CD:1407
- Checkbox object, 615
- concat() method, 251
- constructor (Date object, 286
- Counter object, CD:1767
- CSSMediaRule object, CD:1775
- CSSPrimitiveValue object, CD:1780
- CSSRuleList object, CD:1790
- CSSStyleDeclaration object, CD:1792
- CSSStyleSheet object, CD:1801
- CSSValue object, CD:1805
- CSSValueList object, CD:1807
- Date object, 286
- day, 286, 326
- delete (Array object), 266
- description (Error object), 341
- displaying, code, CD:1962-1963
- Document object, CD:1412, CD:1425
- DOMException object, CD:1429
- DOMImplementation object, CD:1431
- Element object, CD:1434
- Entity object, CD:1445
- Error object, 341-342
- function (Array object), 263-264
- of functions, code to access with arguments array, 79
- hours, 286, 326
- int (Array object), 249
- join() method, 255

label, 280, 283

- location (ActiveXObject), 242
- margins() method, 1017
- milliseconds (UTC() method), 326
- milliseconds1 or milliseconds2 (Date object), 286
- minutes, 286, 326
- month, 286, 326
- Node object, CD:1454
- Notation object, CD:1470
- num (Button object), 597
- num1 or num2 (Checkbox object), 615
- number (Error object), 341
- number of, code to retrieve, CD:1961
- paddings() method, 1021
- passing to displayArguments() function, 79
- ProcessingInstruction object, CD:1472
- push() method, 259
- retrieving, code, CD:1942-1943
- seconds, 286, 326
- serverName (ActiveXObject), 242
- slice() method (Array object), 262
- sort() method (Array object), 263-264
- splice() method (Array object), 266
- start (Array object), 262, 266
- stop (Array object), 262
- stored procedures, code to set, 1268-1269
- string, 255, 286, 597
- typeName (ActiveXObject), 242
- types, stored procedures, 1251
- unshift() method (Array object), 269
- UTC() method (Date object), 326
- value (Boolean object), 273
- year, 286, 326

Arguments object, 243-247**arithmetic operators**

- % (modulus), 47
- * (multiplication), 47
- + (addition), 46, 49-50
- ++ (post-increment), 48
- ++ (pre-increment), 48
- (post-decrement), 49
- (pre-decrement), 48-49
- (subtraction), 46-47
- (unary), 49
- / (division), 47
- Math object, methods, 46
- NaN (Not A Number), 45

arity property, 353, 356

Array object, 42, 249-272

array of items, retrieving, CD:1829

array of keys, retrieving, CD:1830-1831

array subsections, selecting, 262-263

arrayOfNum1 array, 54

arrayOfNum2 array, 54

arrays

[] (square brackets) operator, 40

anchors property, 642

applet, 116-118

applets property, 644

arguments, 79

Array object, 42

arrayOfNum1, 54

arrayOfNum2, 54

as objects, 43-44

document.all property, 639

elements, 129-130, 255-265, 269-270

embeds, 697

food orders, 266

Form, 125-128

form, 762-763

forms property, 657-658

history property, 1117-1118

images property, 664-665

indexes, strings, 41

JavaScript 1.0, 44-45

JScript, converting to Visual Basic

arrays, CD:1829

layers property, 667-668

length, 42

link property, 670-671

multidimensional, 39, 42-43

one-dimensional, 40

options property, 988-98

plug-ins, 118-120

properties, code to access, 44

string indexes, 40

subsections, code to select, 262-263

toString() method, 268-269

Visual Basic, converting to JScript

arrays, CD:1829

ASCII, equivalents, converting hexadecimal values, code, 380

asin() method (Math object), 408, 411-412

ASPs (Active Server Pages), 24-25, 168-169 177

ASPCode property (ASPErrror object), 1183-1184

ASPDescription property (ASPErrror object), 1183-1184

ASPErrror object, 158

ASPErrror object, 168, 1183-1184

ASPErrror.ASPCode property, 1184

ASPErrror.ASPDescription property, 1184

ASPErrror.Category property, 1185

ASPErrror.Column property, 1185

ASPErrror.Description property, 1185-1186

ASPErrror.File property, 1186

ASPErrror.Line property, 1186

ASPErrror.Number property, 1187

ASPErrror.Source property, 1187

assigning

client object properties, 1194-1195

events, CD:1799

methods, 258-259, 275-276

values, 54

variables, 37-38

assignment operator (=), 37, 232

assignment operators, 50, 53-54

associations, ECMA (European Computer Manufacturers Association), 7

asterisk (*) (pattern matching character), 87

AT TARGET constant (Event object), CD:1733

atan() method (Math object), 408, 412-413

atan2() method (Math object), 408, 413-414

atEnd() method (Enumerator object), 337-338

AtEndOfLine property (TextStream object), CD:1912-1914

AtEndOfStream property (TextStream object), CD:1913-1915

atob method (Window object), 1085, 1091

Attr object, CD:1403-1404

Attr.name property, CD:1404

Attr.ownerElement property, CD:1406

Attr.specified property, CD:1404-1405

Attr.value property, CD:1405

attrChange property (MutationEvent object), CD:1756-1758

ATTRIBUTE NODE constant (Node object), CD:1454

attributes

adding, CD:1415

cookies, 139

createAttribute() method, CD:1414

expires, 137

getAttribute() method, CD:1435-1436, CD:1443

getAttributeNode() method, CD:1436

getAttributeNodeNS() method,
CD:1437
getAttributeNS() method, CD:1437
href (tag), 1190
language, 33, 57
length (Array object), 42
method, GET or POST, 124-125
names, 147, CD:1404
path, cookies, 137
patterns, 88
property, CD: 1455-1456
removeAttribute() method, CD:1440
removeAttributeNode() method,
CD:1441
removeAttributeNS() method, CD:1441
retrieving, CD:1846-1847
secure, cookies, 137
setAttribute() method, CD:1442
setAttributeNode() method,
CD:1442-1443
setAttributeNodeNS() method, CD:1443
setAttributeNS() method, CD:1444
specifying, CD:1405
type, 32
values, code to write, CD:1405-1406

**Attributes property, CD:1845-1847, CD:
1891-1893**

**attrName property (MutationEvent
object), CD:1756-1759**

AUTH PASSWORD variable, 1324

AUTH TYPE variable, 1324, 1386

AUTH USER variable, 1324

automated traffic light, 67-69

Automation objects, 272

AvailableSpace property, CD:1833-1834

**availHeight property (screen object),
967-969**

availLeft property, 969

availTop property, 970

availWidth property, 970

axis property, CD:1674-1677

B

\b escape sequence, 36

back() method, 799-800, 1085, 1092, 1117

**background property, 821-825,
CD:1507-1509**

**backgroundColor property, 649, 662, 688,
995, 998-999**

backgrounds, colors, code to set, 353

backslash (\), 35, 87, 160

**BaseElement object, code to create,
CD:1539-1540**

Bcc property (SendMail object), 1355-1357

**beginTransaction() method, 1199,
1201-1202, 1234-1236**

**below property (Layer object), 821,
825-826**

bgColor property

code, 826-827, CD:1510, CD:1677,
CD:1690, CD:1704

Document object, 636

HTMLBodyElement object, CD:1507

HTMLTableCellElement object,
CD:1674

HTMLTableElement object, CD:1688

HTMLTableRowElement object,
CD:1703

Layer object, 821

modifying, code, 646

window.document property, 1108

Big button object, properties, 71

big() method (String object), 498

binaries, COM servers, CD:1824

binary data, writing to files, 1296-1297

binary large objects (BLOB), 1188

**BinaryRead() method (Request object),
1318-1319**

**BinaryWrite() method (Response object),
1335-1336**

**bindings, DOM Level 2 Core language,
CD:1403**

bitwise operators, 59

AND (&), 60, 210-211

AND (&=), 212-213

NOT (-), 60

NOT (~), 224

OR (|), 60, 221

OR (^), 219

OR (^=), 220-221

OR (|=), 223-224

shift left (<<), 61

shift right with sign(>>), 61

shift right zero fill (>>>), 61

XOR (^), 60

blink() method (String object), 498

BLOB (binary large objects), 1188

blob() function, 1188

blob object, 1189

**blobImage() method (blob object),
1189-1191**

blobLink() method, 1191-1192

blocks, 97-101

**blue property (RGBColor object),
CD:1816**

Blur event, 90

BLUR

event (Event object), 698
property, 721-722

blur() method, 1085

Button object, 598
buttons, code to remove focus, 599-600
check boxes, code to remove focus,
617-618
Checkbox object, 615
code, 747, 780, 923-924, CD:1482,
CD:1589
FileUpload object, 746
Frame object, 779
HTMLAnchorElement object, CD:1481
HTMLInputElement object, CD:1586
HTMLSelectElement object, CD:1659
HTMLTextAreaElement object,
CD:1715
Password object, 922
Radio object, 938
Reset object, 952
select boxes, code to remove focus, 977
Select object, 975
Submit button, code to remove focus,
1032
Submit object, 1030
text areas, code to remove focus,
1065-1066
text boxes, code to remove focus, 1048
Text object, 1046
Textarea object, 1063
window focus, code to remove,
1092-1093

**body property (HTMLDocument object),
CD:1525**

**Body property (SendMail object), 1355,
1358, CD:1528**

<body> tag, 118, 151, 781

bold() method (String object), 498

Boolean data type, 36

boolean keyword, 272-273

Boolean object, 273-280

**border parameter (blobImage() method),
1190**

border property

code, 809-810, CD:1579, CD:1626,
CD:1691
HTMLImageElement object, CD:1577

HTMLObjectElement object, CD:1624

HTMLTableElement object, CD:1688

Image object, 807

**borderBottomWidth property (Style
object), 649, 662, 688, 995, 1000**

**borderColor property (Style object), 649,
662, 688, 995, 1001**

**borderLeftWidth property (Style object),
649, 662, 688, 995, 1002**

**borderRightWidth property (Style object),
649, 662, 688, 995, 1002-1003**

**borderStyle property (Style object), 649,
662, 688, 995, 1003**

**borderTopWidth property (Style object),
649, 662, 688, 995, 1004**

**borderWidths() method (Style object),
650, 663, 689, 995, 1005**

**bottom property (Rect object),
CD:1812-1813**

**boxes, upload, code to set focus, 748. See
also alert boxes; check boxes**

brackets

{ } (curly), 63

[] (square), 250

brake parts, inventory list, 43

Break at Next Statement command, 103

break keyword, 71-73, 280

break statements, loops, 75

browsers. See Web browsers

**btoa() method (Window object), 1085,
1094**

**bubbles property (Event object),
CD:1733-1735**

**BUBBLING PHASE constant (Event
object), CD:1733**

Buffer property, 1337

buffers, flushing to pages, 1298

build number (JScript), 195

building

e-mail pages, 174

links dynamically, code, 1176

**BuildPath() method (FileSystemObject
object), CD:1864-1866**

built-in controls for Web browsers, 116-120

**Built-In Maximum Database Connections
application setting, 164**

built-in values of numbers, 34-35

Button object, 71, 597-598

**button property (MouseEvent object),
CD:1748-1750**

Button.blur() method, 599

Button.click() method, 600
Button.focus() method, 601
Button.form property, 602
Button.handleEvent() method, 603
Button.name property, 605
Button.onBlur event handler, 606
Button.onClick event handler, 606-607
Button.onFocus event handler, 607
Button.onMouseDown event handler, 608
Button.onMouseUp event handler, 608
Button.type property, 609
Button.unwatch() method, 610
Button.value property, 612
Button.watch() method, 613
buttons

- creating, 598-599
- focus, 599-602
- names, 605-606
- parent, 602-603
- pop-up message boxes, CD:1996
- Reset, 76
- Submit, 1032-1034
- text, code to modify, 612-613
- type, code to access, 609-610

bvalue property (HTMLTextAreaElement object), CD:1715

byte keyword, 281-282

bytes, strings, 1277-1278, 1291-1292

byteToString() method (File object), 1275-1278

C

-c command-line parameter, 160

C drive, CD:1834, CD:1839-1840

C programming, /* */ comment tags, 33

C subfield (ClientCertificate collection), 1320

C++ programming, // comment tags, 33

CacheControl property, 1338

call by reference (functions), 77

call by value (functions), 77-78

call by value versus call by reference (functions), 78

call() method, 353, 357, 397-399

callC() function, 1192-1193

called checkNum function, 136

callee property, 243-246, 353, 358

caller property, 243, 246-247, 353, 358-359

cancelable property (Event object), CD:1733-1736

canceling events, 93-95

caption property (HTMLTableElement object), CD:1688-1691

captureEvents() method, 647-648

- code, 827-828

- Document object, 637

- Layer object, 822

- window.document property, 1108

- Window object, 1085

capturing events, 92-93, 684-685

CAPTURING PHASE constant (Event object), CD:1733

Car object, 398, 402-405

carriage returns, writing to files, 1297-1298

case keyword, 282

catch block, 97

catching runtime errors, code, 99-100

Category property (ASPErrors object), 1183-1185

Cc property (SendMail object), 1355, 1358

CDATA SECTION NODE constant (Node object), CD:1454

CDATA sections, CD:1415, CD:1419

CDATASection object, CD:1406-1407

ceil() method, code, 414-415

ceil property (Math object), 408

cellIndex property

- (HTMLTableCellElement object),

- CD:1674, CD:1678

cellPadding property (HTMLTableElement object), CD:1688, CD:1692

cells property (HTMLTableRowElement object), CD:1703-1705

cellSpacing property (HTMLTableElement object), CD:1688, CD:1692-1693

CERT COOKIE variable (ServerVariables collection), 1324

CERT FLAGS variable (ServerVariables collection), 1324

CERT ISSUER variable (ServerVariables collection), 1325

CERT KEYSIZE variable (ServerVariables collection), 1325

CERT SECRETKEYSIZE variable (ServerVariables collection), 1325

CERT SERIALNUMBER variable (ServerVariables collection), 1325

CERT SERVER ISSUER variable (ServerVariables collection), 1325

CERT SERVER SUBJECT variable (ServerVariables collection), 1325

CERT SUBJECT variable

(ServerVariables collection), 1325

Certificate field (ClientCertificate collection), 1319**CGIs (common gateway interfaces), 5****ch property**

code, CD:1678, CD:1685,

CD:1705-1706, CD:1711

HTMLTableCellElement object,
CD:1674HTMLTableColElement object,
CD:1684HTMLTableRowElement object,
CD:1703HTMLTableSectionElement object,
CD:1710**Change event, 90****CHANGE**

event (Event object), 698

property, code to access, 722-723,
744-745**changeState() function, 152****char keyword, 282****Character property (WshRemoteError object), CD:1979-1980****CharacterData object, CD:1407****CharacterData.appendData() method, CD:1409****CharacterData.data property, CD:1408****CharacterData.deleteData() method, CD:1411****CharacterData.insertData() method, CD:1410****CharacterData.length property, CD:1408****CharacterData.replicateData() method, CD:1411****CharacterData.substringData() method, CD:1409****characters**

comments, //, 33

literal, 87

/n, 1290

pattern matching, 86-87

/r, 1290

special (strings), 35-36

unencoded, 334-335, 369-370

variables, 37

charAt() method (String object), 498**charCodeAt() method (String object), 498****Charset property, code, 1338, CD:1483, CD:1610, CD:1656****charset property, CD:1480, CD:1609, CD:1655****check boxes**alert boxes, code to display, 627-628
checked properties, code to modify,
618-620

click() method, code, 620-621

creating, code, 615-617

focus, code, 617-618, 623

names, code to display, 626-627

parent access, code, 624

passing events to, code, 625

resetting, code, 621-622

type properties, code to access, 631-632

value properties, code to access, 634-635

Checkbox object, 614-615**Checkbox object:num1 argument, 615****Checkbox object:num2 argument, 615****Checkbox.blur() method, 617****Checkbox.checked property, 618****Checkbox.click() method, 620****Checkbox.defaultChecked property, 621****Checkbox.focus() method, 622****Checkbox.form property, 623-624****Checkbox.handleEvent() method, 624****Checkbox.name property, 626****Checkbox.onBlur event handler, 627****Checkbox.onClick event handler, 628-629****Checkbox.onFocus event handler, 630****Checkbox.type property, 631****Checkbox.unwatch() method, 632****Checkbox.value property, 634****Checkbox.watch() method, 635****checked property, 615, 938, 941, CD:1585, CD:1590****checkPhone function, 136****cheese, code to select, 630-631****child code, 139-141, 144-145****childNodes property (Node object), CD:1455-1458****children**

appendChild() method, code, CD:1457

hasChildren() method, code, CD:1417
objects, 8

removeChild() method, code, CD:1469

replaceChild() method, code, CD:1470

chOff propertycode, CD:1679, CD:1686, CD:1706,
CD:1712HTMLTableCellElement object,
CD:1674

- HTMLTableColElement object,
CD:1684
- HTMLTableRowElement object,
CD:1703
- HTMLTableSectionElement object,
CD:1710
- chromes, AOL and Internet Explorer
browsers, 108**
- cite property, code, CD: 1622-1623,
CD: 1653-1654**
- class keyword, 282**
- classes**
 - creating, 648
 - property, 636, 649-650
 - style sheets, code to create, 650
- className property, 460, CD:1539-1540**
- clear property, 649, 662, 688, 995, 1006,
CD:1513**
- Clear() method (Response object), 1335,
1339**
- clearError() method (File object), 1275,
1278-1279**
- clearInterval() method, 779-781, 1085,
1097**
- clearTimeout() method access, 779-781,
1085, 1098**
- CLICK**
 - event (Event object), 698
 - property, code to access, 723-724
- Click event, 90**
- click events, 600-605, 672**
- click() method**
 - Button object, 598
 - Checkbox object, 615
 - code, 600-601, 620-621, 942, 954-955
 - HTMLInputElement object, CD:1586
 - Radio object, 938
 - Reset object, 952
 - Submit object, 1030
- clicks**
 - handleEvent() method, 1036-1037
 - on pages, 1052-1053, 1070-1071, 1116
- client object, 1193-1197**
- Client Object Maintenance application set-
ting, 165**
- client-side**
 - browser syntax, 563
 - JavaScript, 11-12
 - scripting
 - cookies, 136-139
 - dynamic documents, 145-154
 - form processing, 124-136
 - Web browsers, 108-123
 - windows, 139-145
 - XUL (XML-based User Interface
Language), 154-156
 - server-side, 158-159
 - validation, 131-135
 - versatility, 24-25
- Client-Side JavaScript, Netscape DevEdge
Web site, 21**
- client.destroy() method, 1195**
- client.expiration() method, 1196**
- client.property property, 1196**
- client.unwatch() method, 1197**
- client.watch() method, 1198**
- ClientCertificate collection (Request
object), 1318-1320**
- clientX property (MouseEvent object),
CD:1748-1750**
- clientY property (MouseEvent object),
CD:1748-1751**
- clip.bottom property (Layer object), 821,
828-829**
- clip.height property (Layer object), 821,
829-830**
- clip.left property (Layer object), 821,
830-831**
- clip.right property (Layer object), 821,
831-832**
- clip.top property (Layer object), 821,
832-833**
- clip.width property (Layer object), 821,
833-834**
- cloneNode() method (Node object),
CD:1455, CD:1458**
- close() method**
 - code, 1099-1100, CD:1529, CD:1915
 - Cursor object, 1219
 - cursors, code to close, 1221-1222,
1389-1390
 - Document object, 637
 - document output streams, code to close,
651-652
 - File object, 1275
 - files, code to close, 1279
 - HTMLDocument object, CD:1525
 - Resultset object, 1345
 - resultsets, code to close, 1346-1347
 - Stproc object, 1388
 - TextStream object, CD:1912
 - Window object, 1085
 - window.document property, 1108

closed property (Window), 1101-1102, 1087
closing

cursors, code, 1221-1222, 1389-1390
document output streams, code, 651-652
files, code, 1279
resultsets, code, 1346-1347

clothing stores, displaying inventory, 41
CN subfield (ClientCertificate collection), 1320**code**

, (comma), 215
?: (conditional operator), comparing
with if statement, 59, 218-219
-- (decrement operator), 204-205
() (parentheses), 61
* (multiplication operator), 213
*= (multiplication assignment operator),
214
% (modulus operator), 209
%= (modulus assignment operator),
209-210
! (logical NOT operator), 205-206
!= (not equal operator), 207
!= (non-identity operator), 208
& (bitwise AND operator), 210-211
&= (bitwise AND assignment operator),
212-213
&& (logical AND operator), 211-212
+ (addition operator), numeric and string
values, 49-50
++ (increment operator), 226-227
+= (addition assignment operator), 228
- (subtraction operator), 202-203
- (unary negation operator), 203-204
-= (subtraction assignment operator),
233
/ (division operator), 215
/= (division assignment operator), 217
/* / (multiline comments), 216
// (single-line comments), 216-217
= (assignment operator), 232
== (equal operator), 234
=== (identity operator), 235
A drive readiness, determining, CD:1838
<a> tag, formatting, 1192
Abandon() method, 1380
abbr property, CD:1675
ABORT property, 720-721
above property, 823-824
abs() method, 409-410
accept property, setting, CD:1587

acceptCharset property, CD:1547-1548
accessKey property, CD:1481-1482,
CD:1498, CD:1514, CD:1587-1588,
CD:1603
acos() method, 410-411
action property, CD:1548
ActiveXObject object, creating Excel
multiplication table, 242
add() method, CD:1660-1661, CD:1909
addClient() and redirect() functions,
1176
addEventListener() method,
CD:1745-1746
AddHeader() method, 1335
AddPrinterConnection() method,
CD:1965
addResponseHeader() function, content-
type of files, 1177
AddWindowsPrinterConnection()
method, CD:1966
agent property, accessing, 1328
alert boxes, displaying, 606-609,
627-628, 1041-1042, 1059-1060, 1090
alert() method, 1090
align property, 998, CD:1491-1493,
CD:1523, CD:1564-1566,
CD:1570-1571
aLink property, CD:1508-1509
alinkColor property, setting, 638
ALT MASK modifier, accessing, 707
alt property, CD:1491, CD:1499,
CD:1579, CD:1589
altKey property, CD:1749
Anchor objects, creating, 564-565
Anchor.name property, accessing,
566-567
Anchor.text property, accessing, 567-568
Anchor.unwatch() method, 568-569
Anchor.watch() method, 569-570
Anchor.x property, accessing, 570-571
Anchor.y property, accessing, 571
anchors length property, creating hyper-
links, 642-643
anchors property array, accessing anchor
names, 642
AppActivate() method, CD:1991
appName property, 905
appendChild() method, CD:1457
appendData() method, CD:1410
AppendToLog() method, 1336
Apple Macintosh systems, alerting when
on, 196

- Applet objects, accessing methods, 572
- applets, accessing, 645
- applets property array, accessing applets, 644
- Application object, 1178
- Application.Application_OnEnd event function, 1178
- Application.Application_OnStart event function, 1179
- Application.Contents collection, 1180
- Application.Contents.Lock() method, 1181-1182
- Application.Contents.Remove() method, 1180
- Application.Contents.RemoveAll() method, 1181
- Application.StaticObjects collection, 1182
- Application.Unlock method, 1182-1183
- applications, locking, 1305
- apply() method, 354
- appName property, 906
- appVersion property, 906-907
- archive property, CD:1492, CD:1626
- Area object, passing events, 577-578
- areas, creating, 574-576
- areas property, CD:1616
- arguments, CD:1942-1943, CD:1961-1963
- arguments property, 355, CD:1930-1931, CD:1942-1943, CD:2006
- Arguments property, command-line parameters, CD:1931
- arity property, 356
- Array object, 250-251, 267-268
- arrays
 - creating in JavaScript 1.0, 45
 - elements, 255-257, 260-265, 269-270
 - items, retrieving, CD:1829
 - keys, retrieving, CD:1830-1831
 - multidimensional, 42-43
 - properties, accessing, 44
 - subsections, selecting, 262-263
 - toString() method, 268-269
- asin() method, 411-412
- ASP environment, testing, 169
- ASPCode property, 1184
- ASPDescription property, 1184
- ASPErrors object, 1184
- assignment operators, 51-52
- assignments by values versus reference, 54
- atan() method, 412-413
- atan2() method, 413-414
- atEnd() method, 337-338
- AtEndOfLine property, CD:1913-1914
- AtEndOfStream property, CD:1914-1915
- atob() method, 1091
- attrChange property, CD:1758
- attributes, 1404-1406, CD:1415, CD:1846-1847
- Attributes property, CD:1456, CD:1893
- attrName property, CD:1758-1759
- automated traffic light, 67-69
- availHeight property, accessing, 968-969
- availLeft property, accessing, 969
- availTop property, accessing, 970
- availWidth property, accessing, 970
- axis property, CD:1676-1677
- back() method, 800, 1092
- background colors, setting, 353
- background property, 824-825, CD:1509
- backgroundColor property, 998-999
- backgroundImage property, 999-1000
- BaseElement object, creating, CD:1539-1540
- Bcc property, 1357
- beginTransaction() method, 1201-1202, 1236
- below property, 825-826
- bgColor property, 646, 826-827, CD:1510, CD:1677, CD:1690
- BinaryRead() method, 1319
- BinaryWrite() method, 1336
- BLOB data, inserting into databases, 1188
- blob() function, inserting BLOB data into databases, 1188
- BLUR property, 721-722
- blur() method, 747, 780, 953-954, CD:1482, CD:1589
 - buttons, removing focus, 599-600
 - check boxes, code to remove focus, 617-618
 - select boxes, code to remove focus, 977
 - Submit button, code to remove focus, 1032
 - text areas, removing focus, 1065-1066

- text boxes, removing focus, 1048
- window focus, removing, 1092-1093
- blur property, reading, CD:1816
- Body property, 1358, CD:1528
- Boolean object versus primitive Boolean value, 274
- border property, 809-810, CD:1579, CD:1626, CD:1691
- borderBottomWidth property, 1000
- borderColor property, 1001
- borderLeftWidth property, 1002
- borderRightWidth property, 1002-1003
- borderStyle property, 1003
- borderTopWidth property, 1004
- borderWidths() method, 1005
- bottom property, reading, CD:1813
- break keyword, labels, 280-281
- breaks and labels (loops), 72-73
- btoa() method, 1094
- bubbles property, CD:1735
- Buffer property, 1337
- Buffers, flushing to pages, 1298
- button focus, shifting, 601-602
- Button object property names, accessing with for...in loop, 71
- Buttons, 590-606, 609-613, CD:1750
- bytes into strings, converting, 1277-1278
- byteToString() method, 1277-1278
- C drive, CD:1834, CD:1839-1840
- CacheControl property, 1338
- call by value (functions), 77
- call() method, 357, 398-399
- callC() function (registered), 1192-1193
- callee property, 244-246, 358
- caller property, 246-247, 358-359
- cancelable property, CD:1735-1736
- caption property, CD:1689, CD:1691
- captureEvents() method, 647-648, 827-828
- Car object, 398, 402-405
- Category property, 1185
- Cc property, 1358
- CDATA sections, CD:1415, CD:1419
- ceil() method, 414-415
- cellIndex property, CD:1678
- cellPadding property, CD:1692
- cells property, CD:1705
- cellSpacing property, CD:1692-1693
- ch property, CD:1678, CD:1685, CD:1705-1706, CD:1711
- CHANGE property, accessing, 722-723, 744-745
- Character property, CD:1980
- Charset property, 1338, CD:1483, CD:1610, CD:1656
- check boxes
 - checked properties, code to modify, 618-620
 - click() method, 620-621
 - creating, 615-617
 - focus, applying, 623
 - focus, code to remove, 617-618
 - names, code to display, 626-627
 - parent access, 624
 - passing events to, 625
 - resetting, 621-622
 - type properties, code to access, 631-632
 - value properties, code to access, 634-635
- checked property, 941, CD:1590
- cheese, selecting, 630-631
- child, windows, 144-145
- childNodes property, iterating, CD:1457-1458
- chOff property, CD:1679, CD:1686, CD:1706, CD:1712
- cite property, CD:1623, CD:1654
- className property, CD:1540
- Clear() method, 1339
- clear property, 1006, CD:1513
- clearError() method, 1278-1279
- clearInterval() method, 780-781, 1097
- clearTimeout() method, 781, 1098
- click events, 600-605, 672
- click() method, 600-601, 620-621, 954-955, 1033, CD:1590-1591
- CLICK property, accessing, 723-724
- clicks on pages, 1052-1053, 1070-1071
- client object properties, 1194-1197
- ClientCertificate collection, 1320
- clientX property, CD:1750
- clientY property, CD:1751
- clip.bottom property, 828-829
- clip.height property, 829-830
- clip.left property, 830-831
- clip.right property, 831-832
- clip.top property, 832-833
- clip.width property, 833-834
- cloneNode() method, CD:1458
- close() method, 1099-1100, CD:1529, CD:1915

- cursors, closing, 1221-1222, 1389-1390
- document output streams, closing, 651-652
- files, closing, 1279
- resultsets, closing, 1346-1347
- closed property, 1101-1102
- code letters, displaying, CD:1835
- code property, CD:1627
- codeBase property, CD:1493, CD:1627-1628
- CodePage property, 1380
- codeType property, CD:1628
- color parameter, displaying, CD:1948-1949
- color property, 1007, CD:1506, CD:1545
- colorDepth property, accessing, 971
- cols property, CD:1561, CD:1717
- colSpan property, CD:1679
- column names (tables), 1223-1224
- Column property, 1185, CD:1916
- columnName() method, 1223-1224, 1347-1348
- columnName property, 1222
- columns() method, 1224, 1348-1349
- comments, 17, CD:1416
- commitTransaction() method, 1202-1203, 1237
- compact property, CD:1522-1525, CD:1618, CD:1637, CD:1726
- compile() method, 476-477
- complete property, 810
- ComputerName property, CD:1966-1967
- conditional compilations, 200
- confirm() method, 1103
- connect() method, 1238-1239, 1259-1264
- connected() method, connections, 1203-1204, 1239-1240, 1260-1261
- Connection object, creating, 1200-1201
- connections, 1203-1204, 1210-1211, 1239-1240
- ConnectObject() method, CD:1932
- constructor property, 253, 275, 359, 453-454, 1280, 1301-1302
- content property, CD:1619
- content-type of files, setting, 1177
- contentDocument property, CD:1555, CD:1571, CD:1629
- Contents collection, 1381
- ContentType property, 1339
- contextual() method, changing italic text, 652
- continue statement, 284-285
- continue statement and labels, 74
- cookies, 138-139, 653, CD:1529
- Cookies collection, 1321, 1340
- coords property, CD:1483, CD:1499
- Copy() method, CD:1894
- cos() method, 415-416
- Count() method, CD:1946, CD:1956, CD:1961, CD:2014, CD:2017-2018
- Count property, 1322-1324, CD:1910-1911
- Counter object, properties, CD:1767-1768
- createAttribute() method, CD:1414
- createCaption() method, CD:1693
- createCSSStyleSheet() method, invoking, CD:1811
- createDocument() method, CD:1432
- createDocumentFragment() method, CD:1416-1417, CD:1425
- createDocumentType() method, CD:1433
- createEvent() method, CD:1732-1734
- createHTMLDocument() method, CD:1538
- CreateObject() method, 1369, CD:1933, CD:1953-1954
- CreateShortcut() method, CD:1992
- createTextNode() method, CD:1420
- createTFoot() method, CD:1694
- createTHead() method, CD:1694
- crypto property, 1103-1104
- crypto.random() method, 1104
- crypto.signText() method, 1105
- CSSRule object, CD:1787
- CSSRuleList object, iterating, CD:1790, CD:1794, CD:1806, CD:1817
- cssRules property, iterating, CD:1775-1776, CD:1801-1802
- CSSStyleSheet object, creating, CD:1801
- cssText property, CD:1788, CD:1793-1794, CD:1806
- CSSValueList object, iterating, CD:1807
- cssValueType property, reading, CD:1806
- ctrlKey property, CD:1751-1752
- current property, 801
- CurrentDirectory property, CD:1992-1993

- currentTarget property, CD:1736
- cursor() method, 1204, 1240
- Cursor object, 1220-1221
- cursors, 1067-1068, 1221-1222, 1389-1390
- D drive, finding, CD:1873
- Data, decoding, 1091, 1094
- data property, CD:1408, CD:1473
- database connection errors, 1205-1208, 1243-1245, 1264-1267
- database connections, disconnecting, 1241-1242
- database object, creating, 1234-1235
- database.prototype property, 1246-1247
- databases, 1204, 1238-1240, 1259-1264
- Date objects, 288, 323
- DateCreated property, CD:1894-1895
- DateLastAccessed property, CD:1895-1896
- DateLastModified property, CD:1896-1897
- dates, 321-327
- dateTime property, CD:1623
- DbClick events, 673
- DBLCLICK property, accessing, 724-725
- DbPool instances, creating, 1262-1263
- DbPool() method, code, 1262-1263
- DbPool object, creating, 170, 1257-1258
- DbPool.prototype property, 1267-1268
- debug() function, 1273
- debugging with alert() method, 105-106
- DEC Alpha processor, detecting, 194
- declare property, CD:1630
- decodeURI() function, 329, 367
- decodeURIComponent() function, 330-331, 368
- defaultChecked property, 942-943, CD:1591
- defaultSelected property, 916, CD:1642
- defaultStatus property, 1106
- defaultValue property, 924-925, 1066-1067, 1049, CD:1592, CD:1717-1718
- defer property, CD:1656
- Delete() method, CD:1897
- delete operator, 331-332
- deleteCaption() method, CD:1695
- deleteCell() method, CD:1707
- deleteData() method, CD:1411
- deleteResponseHeader() function, 1274
- deleteRow() method, 1225-1226, CD:1695-1696, CD:1712
- deleteRule() method, CD:1776, CD:1802
- deleteTFoot() method, CD:1696
- deleteTHead() method, CD:1697
- description property, 342-343, 935, 1186, CD:1981, CD:2006-2007
- destroy() method, 1195
- detail property, CD:1762
- dictionaries, items, CD: 1826-1828
- Dictionary object, CD:1825-1832
- disabled property, CD:1515, CD:1592-1593, CD:1611, CD:1640, CD:1662
- disableExternalCapture() method, 1106-1107
- disconnect() method, 1241-1242
- dispatchEvent() method (EventTarget object), CD:1744-1746
- display property, 1008
- displayArguments() function, 79
- DML queries, running, 1205, 1242
- docType property, checking, CD:1420-1421
- document anchors, iterating, CD:1526-1527
- document applets, iterating, CD:1527-1528
- document layers, displaying, 668-669
- Document object, creating, CD:1413, CD:1435
- document property, 781-784, 834-835
- document.all property array, 639
- documentElement property, CD:1421
- documents, 680, CD:1418
- DOM (Document Object Model), uses of, 149-151
- domain property, CD:1530
- DOMException object, CD:1430-1431
- DOMImplementation object, CD:1431-1432
- do...while loop, 333
- DRAGDROP property, 726
- drive letters, displaying, CD:1835
- Drive object, CD:1833-1835
- Drive property, CD:1898
- drives
 - accessing, 338-339
 - checking, CD:1874
 - moving to, 340-341

- number of, accessing, CD:1843
- retrieving, CD:1844-1845
- share names, returning, CD:1841
- total size, displaying, CD:1841-1842
- types, displaying, CD:1836
- volume name, returning, CD:1842
- Drives collection, displaying number of items, CD:1843-1844
- E property, 416-417
- Echo() method, CD:1934
- elements (Document object), creating, CD:1435
- elements property, CD:1549
- elements.length property, 765-766
- else keyword, 334
- embedded objects, 654-655, 697
- embeds property, listing embedded objects, 654
- enabledPlugin property, accessing, 901
- enableExternalCapture() method, 1111
- encodeURIComponent() function, 335, 369
- encodeURIComponent() function, 336, 370
- encoding property, reading, CD:1770-1771
- enctype property, CD:1550
- End event, CD:1973-1974
- End() method, 1340
- entities property, CD:1426
- entity references, creating, CD:1418-1419
- Enumerator object, creating, 337
- enumerators, moving, 339-340
- EnumNetworkDrives() method, retrieving network drive mapping, CD:1967
- EnumPrinterConnection() method, displaying printers, CD:1968
- Environment property, CD:1993
- environment variables, 1387, CD:1956-1959
- eof() method, finding end of files, 1280-1281
- equals() method, 400
- Error event, CD:1974
- error() method, 1281-1282
- Error object, creating, 342
- Error property, CD:1975
- ERROR property, accessing, 727
- error codes, displaying, 1359-1360
- Errorsto property, 1361
- escape() function, 345-346
- escape() method, 371
- eval() function, 347, 372
- eval() method, 400-401, 454
- event capturing, preventing, 684-685
- event event, 1384
- event handlers, defining with dot notation, 93
- Event keyword, 699-700
- event.data property, accessing, 701-702
- event.height property, accessing, 702-703
- event.layerX property, accessing, 703-704
- event.unwatch() method, 715-716
- EventException object, functions, CD:1741-1743
- EventListener object, functions, CD:1743
- eventPhase property, CD:1737
- events, 577-578, 660-661, CD:1799
- exec() method, 477-478
- Exec() method, CD:1994
- Execute() method, 1370, CD:1976
- execute() method, 1205, 1242
- exists() method, files, checking for existence, 1282
- exp() method, 417-418
- ExpandEnvironmentStrings() method, CD:1994-1995
- expiration() method, 1196
- Expires property, 1341
- ExpiresAbsolute property, 1341
- export keyword, 348
- face property, CD:1506, CD:1545
- favorite foods, displaying, 243-244
- fgColor property, text colors, 656
- File object, 1276-1277, CD:1845-1846, CD:1856-1857
- File property, 1186
- filename property, 935-936
- filenames, retrieving, CD:1854
- files
 - binary data, writing, 1296-1297
 - checking, 1282, CD:1875
 - closing, 1279
 - contents, displaying, 175-176
 - copying, CD:1848, CD:1867
 - creation dates, retrieving, CD:1848-1849
 - dates last accessed, code, CD:1849-1850

- dates last modified, CD:1850-1851
 - deleting, CD:1851-1852, CD:1871
 - drive letters, obtaining, CD:1852-1853
 - errors, clearing, 1278-1279
 - finding end of, 1280-1281
 - moving to new directory, CD:1853
 - opening, 1286
 - short names, accessing, CD:1858
 - short paths, obtaining, CD:1858-1859
 - sizes, obtaining, CD:1859-1860, CD:1882
 - strings, 1295-1296, CD:1855-1856
 - systems, displaying, CD:1836-1837
 - types of, CD:1860
 - writing strings and carriage returns, 1297-1298
- Files object, CD:1861-1864
- Files property, CD:1899
- FileSystemObject object, CD:1865-1870, CD:1877-1883, CD:1886-1890
- FileUpload object, 746-747
- FileUpload.form property, 749-750
- FileUpload.name property, accessing, 752-753
- FileUpload.onBlur event handler, 753-754
- FileUpload.type property, accessing, 757
- FileUpload.value property, 759
- find() method, 1112
- firstChild property, reading, CD:1459
- floor() method, 418-419
- Flush() method, 1342
- flush() function, buffers, 1298
- flush() method, 1283
- focus, setting to upload box, 748
- focus() method, 925, 943-944, 1112-1113, CD:1484, CD:1593
 - accessing, 782
 - buttons, shifting focus, 601-602
 - select boxes, code to place cursor, 978-979
 - Submit button, code to set focus, 1034
 - text areas, placing cursor, 1067-1068
 - text boxes, placing cursor, 1050-1051
- FOCUS property, accessing, 728-729
- Folder object, CD:1892
- folder objects, retrieving, CD:1884
- folders, deleting, CD:1872
- Folders collection, CD:1909
- fontFamily property, 1008-1009
- fontSize property, 1010
- fontStyle property, 1010-1011
- fontWeight property, 1011-1012
- food orders, simulating, 266
- for loop, 66, 350
- Form collection, 1322
- Form object, 658-659, 761-764
- form property, 794, CD:1543-1544, CD:1593-1594, CD:1624-1625, CD:1719
 - buttons, accessing parent, 602-603
 - forms, accessing, 1051-1052, 1069
 - forms, code to access, 979-980
- Form.encoding property, accessing, 766-767
- Form.length property, 768-769
- formKey property, reading, 1329
- forms
 - accessing, 126-127, 979-980, 1035-1036, 1051-1052, 1069
 - client-side validation, 131-135
 - elements, accessing, 129, 764-765
 - iterating, CD:1530-1531, CD:1540-1541
 - names, accessing, 657-658
 - submitting, 1033
 - writing information, 130-131
- forms actions, accessing, 656-657
- forward() method, 801-802, 1114, 1171-1172
- for...in object, 351
- Frame object, 779
- frame property, CD:1697
- Frame.parent property, 787
- frameBorder property, CD:1556, CD:1572
- frames property, 783, 1114
- free space, retrieving, CD:1837-1838
- From property, 1362

fruit dictionaries, CD:1825-1827

FullName property, CD:1934, CD:2007, CD:2021

function arguments, accessing with arguments array, 79

function keyword, 352

getAttribute() method, CD:1435-1436, CD:1443

getAttributeNode() method, CD:1436

getAttributeNodeNS() method, CD:1437

getAttributeNS() method, CD:1437

getComputedStyle() method, CD:1819

getCounterValue() method, CD:1782

getDate() method (Date object), 289-290

getDay() method (Date object), 290-291

GetDrive() method (FileSystemObject object), CD:1833-1834

getElementsByName() method, CD:1531-1532

getElementsByTagName() method, CD:1438

getElementsByTagNameNS() method, CD:1438-1439

getFloatValue() method, CD:1782-1783

getFullYear() method (Date object), 291-292

getHours() method (Date object), 292

GetLastError() method, 1370

getLength() method, 1284

getMilliseconds() method (Date object), 293

getMinutes() method (Date object), 293

getMonth() method (Date object), 294-295

getNamedItem() method, CD:1448-1449

getNamedItemNS() method, CD:1449

GetObject function, 365

getOptionValue() function, 1299-1300

getOptionValueCount() function, 1300

getOverrideStyle() method, overriding, CD:1809-1810

getPosition() method, 1285

getPropertyCSSValue() method, invoking, CD:1794

getPropertyPriority() method, CD:1795

getPropertyValue() method, CD:1795-1796

getRectValue() method, CD:1783

getRGBColor() method, CD:1781-1782

getRGBColorValue() method, CD:1784

getSeconds() method (Date object), 295-296

getSlot() method, 403

getStringValue() method, CD:1784

getTime() method (Date object), 296-297

getTimezoneOffset() method (Date object), 297

getUTCDate() method (Date object), 298

getUTCTDay() method (Date object), 298-299

getUTCFullYear() method (Date object), 300

getUTCHours() method (Date object), 300-301

getUTCMilliseconds() method (Date object), 301

getUTCMinutes() method (Date object), 302

getUTCMonth() method (Date object), 302-304

getUTCSeconds() method (Date object), 304

getVarDate() method (Date object), 305

getWindow() method, 403

getWrappedException() method, 396

getYear() method (Date object), 305-306

global property, 478

go() method, 802

green property, reading, CD:1817

grocery inventory, displaying, 251-252

handleEvent() method, 603-605, 751-752, 836, 981-982, 1036-1037, 1116

clicks on pages, 1052-1053,

1070-1071

events, passing, 660-661

hasAttribute() method, CD:1439

hasAttributeNS() method, CD:1440

hasAttributes() method, CD:1459-1460

hasChildren() method, CD:1417

hasFeature() method, CD:1434

hash property, 579-580, 867-868, 887

headers property, CD:1680

height property, 811-812, 971-972,

CD:1494, CD:1572, CD:1631, CD:1680-1681

- Hello World script, 185-186
- hexadecimal values, converting to ASCII equivalents, 380
- Hidden object, code, 793-794
- History object, 799
- home() method, 1118
- host property, 581, 868-869, 888, 1374
- hostname property, 582-583, 869-870, 889-890, 1375
- Hotkey property, CD:2008
- href property, 583-584, 870-871, CD:1484-1485, CD:1500, CD:1609-1612, CD:1772-1773
- hrefLang property, CD:1485, CD:1612
- hspace property, 812-813, CD:1494, CD:1580-1581, CD:1632
- HTML (Hypertext Markup Language), 122-123, 176, 640-641
- HTMLAnchorElement object, CD:1481-1489
- HTMLAppletElement object, CD:1491-1497
- HTMLAreaElement object, CD:1498-1503
- HTMLBaseElement object, CD:1503-1505
- HTMLBaseFontElement object, CD:1506-1507
- HTMLBodyElement object, CD:1508-1512
- HTMLBRElement object, CD:1512-1513
- HTMLButtonElement object, CD:1513-1518
- HTMLCollection object, CD:1518-1521, CD:1532
- HTMLDirectoryElement object, CD:1521-1522
- HTMLDivElement object, CD:1523
- HTMLDListElement object, CD:1524-1525
- HTMLDocument object, CD:1526
- HTMLDOMImplementation object, CD:1538
- HTMLElement object, CD:1540
- HTMLEncode() method, 1370
- HTMLFieldSetElement object, CD:1543-1544
- HTMLFontElement object, CD:1545-1546
- htmlFor property, CD:1604
- HTMLFormElement object, CD:1547-1554
- HTMLFrameElement object, CD:1554-1560
- HTMLFrameSetElement object, CD:1561-1562
- HTMLHeadElement object, CD:1562-1563
- HTMLHeadingElement, CD:1564-1565
- HTMLHRElement object, CD:1565-1568
- HTMLHtmlElement object, CD:1568
- HTMLIFrameElement object, CD:1570-1571
- HTMLImageElement object, CD:1578-1585
- HTMLInputElement object, CD:1586-1600
- HTMLIsIndexElement object, CD:1600-1602
- HTMLLabelElement object, CD:1602-1604
- HTMLLegendElement object, CD:1605-1607
- HTMLLIElement object, CD:1607-1609
- HTMLLinkElement object, CD:1609-1615
- HTMLMapElement object, CD:1616-1617
- HTMLMenuElement object, CD:1617-1618
- HTMLMetaElement object, CD:1619-1621
- HTMLModElement object, CD:1622-1623
- HTMLObjectElement object, CD:1624-1636
- HTMLOLListElement object, CD:1637-1638
- HTMLOptGroupElement object, CD:1639-1641
- HTMLOptionElement object, CD:1641-1647
- HTMLParagraphElement object, CD:1648-1649
- HTMLParamElement object, CD:1649-1652
- HTMLPreElement object, CD:1652-1653
- HTMLQuoteElement object, CD:1653-1654

HTMLScriptElement object,
CD:1655-1659
HTMLSelectElement object,
CD:1660-1670
HTMLStyleElement object,
CD:1670-1673
HTMLTableCaptionElement object,
CD:1673-1674
HTMLTableCellElement object,
CD:1675-1683
HTMLTableColElement object,
CD:1684-1688
HTMLTableElement object,
CD:1689-1702
HTMLTableRowElement object,
CD:1703-1709
HTMLTableSectionElement object,
CD:1710-1714
HTMLTextAreaElement object,
CD:1715-1724
HTMLTitleElement object,
CD:1724-1725
HTMLUListElement object,
CD:1726-1727
httpEquiv property, CD:1620
hyperlinks, creating, 642-643
IconLocation property, CD:2009
identifier property, reading, CD:1768
identifiers, retrieving elements,
CD:1421-1422
if statement, 382
if...else statement, 63-64, 382-383
ignoreCase property, 479
Image object, 808-809
images, 664-665
images property array, 664-665
imageX property, 1330
imageY property, 1330
 tag, formatting, 1189-1191
implementation property, CD:1423-1424
import keyword, 384
in keyword, 384-385
index property, 253-254, CD:1644
Infinity keyword, 385-386
Infinity property, 373-374
initEvent() method, CD:1737-1738
innerHeight property, 1119
innerWidth property, 1120-1121
input property, 254, 480
insertBefore() method, CD:1461
insertCell() method, CD:1707
insertData() method, CD:1410
insertRow() method, 1226-1227,
CD:1698, CD:1713
insertRule(), CD:1777, CD:1803
instance properties, displaying, 975-976,
1030-1031
instanceof keyword, 386-387
instances of Window object, creating,
1088-1090
Intel processor, alerting when on,
199-200
Interactive property, retrieving script
mode, CD:1936
internalSubset property, reading,
CD:1427
ip property, verifying authorization,
1331
IsClientConnected property, 1342
isFinite() method, 374-375, 388
isMap property, CD:1581
isNaN() function, 375-376, 388-389
IsRootFolder property, CD:1900
isSupported() method, CD:1461-1462
isValid() method, 1302
italic text, changing, 652
item() method, CD:1450
drives, accessing, 338-339
HTML elements, code to find,
640
invoking, CD:1796,
CD:1807-1808
iterating, CD:1791
Item() method, CD:1946-1947,
CD:1957
Item property, CD:1911-1912,
CD:1962-1963, CD:2014-2015,
CD:2018
Java, 116-117, 121-122
java object, 389-390
javaArray object, 390-391
javaClass object, 393
javaEnabled() method, 907-908
javaObject object, 393-394
javaPackage object, 394-395
JavaScript, loading applet, 401
JSript, 177, 194-195
JSEException object, 395
JSObject object, 397
KeyDown events, 674
KEYDOWN property, accessing,
729-730
KeyPress events, 675

- KEYPRESS object, accessing, 730-731
- keys, changing, CD:1830
- keystrokes, sending to Notepad, CD:2002
- KeyUp events, 676
- KEYUP property, accessing, 731-732
- label keyword, 407
- label property, CD:1640-1641, CD:1645
- lang property, writing, CD:1542
- language property, 908
- lastChild property, reading, CD:1462
- lastIndex property, 481
 - lastIndex property (Array object), 256
- lastMatch property, 482
- lastModified property, displaying prices, 666-667
- lastParen property, 482-483
- Layer object, 667-668, 823
- layers property array, 667-668
- layerY property, 704-705
- LCID property, 1383
- left property, 837, CD:1814
- leftContext property, 483-484
- length property, 360, 783, 1115, CD:1450-1451, CD:1947-1948
 - accessing, 989
 - applets, code to access, 645
 - arguments, displaying, CD:1963
 - embedded objects, listing, 655
 - environment variables, code to display, CD:1958
 - getting, CD:1796-1797
 - images, displaying number of, 665
 - iterating, CD:1520
 - layers in documents, code to display, 668-669
 - links, displaying, 671
 - property (Arguments object), 248-249
 - reading, CD:1791-1792, CD:1808
 - select box names, code to retrieve, 982-983
 - text data length, checking, CD:1408-1409
- Line property, 1186, CD:1916-1917, CD:1981-1982
- lineHeight property, 1012-1013
- link colors, setting, 669-670
- Link object, 865-866
- link property, CD:1510
- link property array, displaying URLs, 670-671
- linkColor property, link colors, 669-670
- links
 - building dynamically, 1176
 - colors, setting, 692-693
 - creating, 683
 - displaying, 671
 - iterating, CD:1533
- listStyle property, reading, CD:1769
- listStyleType property, 1013-1014
- LN10 property, 419
- LN2 property, 420
- Load events, 677
- load() method, 838
- LOAD property, 733
- localName property, assigning, CD:1462-1463
- Location object, 886
- location property, 1122
- locationbar property, 1122
- locationbar.visible property, 1123
- lock() method, 1303, 1313, 1375
- Lock() object, creating, 1301
- Lock.isValid() method, 1302
- locking, 1303, 1313
- locks, verifying, 1302
- log() method, 421
- LOG2E property, 423
- LOG10E property, 422
- LogEvent() method, CD:1995-1996
- longDesc property, CD:1556, CD:1573, CD:1582
- loops, stopping, 337-338
- lowsrc property, 813, CD:1577-1578, CD:1582
- majorErrorCode() method, 1205-1206, 1243, 1264
- majorErrorMessage() method, 1206-1207, 1243-1244, 1265
- MapNetworkDrive() method, CD:1968-1969
- MapPath() method, 1371
- marginBottom property, 1014-1015
- marginHeight property, CD:1557, CD:1573-1574
- marginLeft property, 1015
- marginRight property, 1016
- margins() method, 1017
- marginTop property, 1018

- marginWidth property, CD:1557-1558, CD:1574
- matches, displaying positions, 253-256
- Math object, creating, 409
- max() method, 423-424
- MAX VALUE property, 441-442
- maxLength property, CD:1594
- media property, CD:1612-1613, CD:1672, CD:1774, CD:1777
- media types, setting, CD:1775
- menubar property, 1124
- menubar.visible property, 1124-1125
- message property (Error object), displaying, 343
- messages, displaying, 629-630
- metaKey property, CD:1753
- method property, 769-770, 1331-1332, CD:1551
- methods
 - assigning, 258-259, 275-276
 - creating, 307-308, 1209-1210, 1228-1229, 1246-1247, 1267-1268
- MimeType object, 899-900
- mimeTypes property, 909
- min() method, 424-425
- MIN VALUE property, 442
- minorErrorMessage() method, 1266-1267
- minorErrorCode() method, 1207-1208, 1244-1245, 1265-1266
- minorErrorMessage() method, 1208, 1245
- modifier values, accessing, 707
- modifiers property, accessing, 706-707
- Motorola 680-0 processors, alerting when on, 197
- Motorola PowerPC processors, alerting when on, 197-198
- mouse events, CD:1748-1749, CD:1752
- MouseDown events, 677-678
- MOUSEDOWN property, accessing, 734
- MOUSEMOVE property, accessing, 735
- MOUSEOUT property, 736
- MOUSEOVER property, accessing, 736-737
- MouseUp events, 678-679
- MOUSEUP property, accessing, 737-738
- Move() method, CD:1900-1901
- MOVE property, accessing, 738-739
- moveAbove() method, 839
- moveBelow() method, 840
- moveBy() method, 840-841, 1125-1126
- moveFirst() method, 339-340
- moveNext() method, 340-341
- moveTo() method, 841-842, 1127
- moveToAbsolute() method, 842-843
- multidimensional arrays, 42-43
- multiline property, 484-485
- multiple property, CD:1664
- multiplication tables, creating, 66
- mutation events, CD:1757, CD:1759
- name property, 814, 927-928, CD:1427, CD:1516, CD:1617
 - accessing, 770-771
 - attributes, writing names, CD:1404
 - buttons, accessing name, 605-606
 - check boxes, displaying, 626-627
 - displaying, 344
 - select box names, code to retrieve, 983-984
 - text area names, retrieving, 1071-1072
 - text boxes, retrieving name, 1054
- Name property, retrieving WSH name, CD:1936
- Named property, displaying color parameter, CD:1948-1949
- namedItem() method, iterating, CD:1521
- namespaceURI property, reading, CD:1463
- NaN object, 376, 436
- NaN property, 443
- navigator object, 904-905
- NEGATIVE INFINITY property, 444
- netscape object, 437
- netscape package, 462-463
- network drive mappings, retrieving, CD:1967
- network printers, displaying, CD:1968
- new operator, 438
- newValue property, comparing with prevValue property, CD:1760-1761
- next() method, 1227-1228, 1349-1350
- next property, 803-804
- nextSibling property, reading, CD:1464
- nodeName property, checking, CD:1464
- nodes, importing, CD:1424

- nodeType property checking, CD:1465
- nodeValue property, processing, CD:1465-1466
- noHref property, CD:1500-1501
- noResize property, CD:1559
- normalize() method, CD:1466
- noShade property, CD:1566-1567
- notationName property, reading, CD:1445-1446
- notations property, reading, CD:1428
- noWrap property, CD:1681
- Number constructor, 440
- Number() method, 377, 439
- Number property, 1187, CD:1982-1983
- number property (Error object), displaying, 345
- Number.valueOf() method, 451
- numbers, adding or concatenating, 225
- Object object, 453
- object property, CD:1495-1496
- ObjectContext object, 1309
- offscreenBuffering property, 1129
- onAbort event handler, 814-815
- onBlur event handler, 844-845, 928-929, 946-947, 1038-1039, 1055, 1130
 - accessing, 784-785
 - alert boxes, code to display, 606
 - alert boxes, displaying, 627-628
- onChange event handler, 754-755, 985-986, 1056, 1073-1074
- onClick event handler, 871, 947-948, 960
 - alert boxes, code to display, 607
 - click events, code, 672
 - messages, displaying, 629-630
 - textbox contents, displaying, 1039-1040
- onDbClick event handler, 585, 673, 872
- onDragDrop event handler, 1130
- onError event handler, 815, 1131
- onFocus event handler, 755-756, 845-846, 948-949, 1040-1041, 1131-1132
 - alert boxes, code to display, 607-608
 - checkbox, selecting, 630-631
- onKeyDown event handler, 674, 815, 872, 1076
- onKeyPress event handler, 675, 816, 873, 1076-1077
- onKeyUp event handler, 676, 816, 874, 1077-1078
- onLoad event handler, 677, 817, 846-847, 1132
- onMouseDown event handler, 677-678, 874-875
- onMouseOut event handler, 585-586, 847-848, 875-876
- onMouseOver event handler, 586-587, 848-849, 876-877
- onMouseUp event handler, 678-679, 877
- onMove event handler, 786, 1132
- onOnMouseDown event handler, 608
- onOnMouseUp event handler, 609
- onReset event handler, 771-772
- onResize event handler, 786-787, 1133
- onSelect event handler, 1058, 1078-1079
- onSubmit() method, 772-773
- OnTransactionAbort event, 1310
- OnTransactionCommit event, 1310
- onUnload event handler, 680, 1133
- open() method, 680, 1136-1137, CD:1534
- opener property, 1138-1139
- Option object, 915
- options property, CD:1665-1666
- options property array, retrieving select box properties, 988-989
- Organization property, 1362
- outerHeight property, 1139-1140
- outerWidth property, 1141
- outParamCount() method, 1390-1391
- outParameters() method, 1391-1392
- ownerDocument property, reading, CD:1467
- ownerElement property, CD:1406
- ownerRule property, reading, CD:1803
- Packages object, 460-461
- Packages.className property, 461
- Packages.java subpackage, 462
- paddingBottom property, 1019
- paddingLeft property, 1019-1020
- paddingRight property, 1020-1021
- paddings() method, 1021-1022
- paddingTop property, 1022-1023
- pages, accessing, 1117-1118
- pageX property, 707-708, 849-850
- pageXOffset property, 1142-1143
- pageY property, 709, 850-851
- pageYOffset property, 1143-1144

- parent folders, code to get names, CD:1885
- parent property, 1144
- parent windows, 142, 144
- ParentFolder property, CD:1902
- parentLayer property, 851-852
- parentNode property, checking, CD:1467
- parentRule property, CD:1788, CD:1797
- parentStyleSheet property, reading, CD:1789
- parse() method (Date object), 306
- parseFloat() method, 377-378, 464
- parseInt() method, 378-379, 465
- Password object, 923
- passwords, validating, 98-101
- Path property, CD:1903, CD:1937
- pathname property, 587-588, 878, 891-892
- pattern matching, regular expressions, 89
- personalbar property, visible property, 1145-1146
- PI property, 426
- PICS property, 1343
- pixelDepth property, accessing, 972
- platform property, 909-910
- plug-ins, displaying, 681-682
- plug-ins array, checking, 118-119
- Plugin object, 934
- plugins property, 681, 910
- plugins.refresh() method, 911
- pools, connections, 171
- Popup() method, CD:1996-1997
- port property, 588-589, 879, 892-893, 1376
- POSITIVE INFINITY property, 444-445
- pow() method, 426-427
- preference() method, 912
- prefix property, reading, CD:1468
- preventDefault() method, CD:1738
- previous property, 804-805
- previousSibling property, reading, CD:1468-1469
- prevValue property, comparing with newValue property, CD:1760-1761
- prices, displaying, 666-667
- primitiveType property, accessing, CD:1785
- print() method, 788, 1146-1147
- printers, adding, 184
- problem, debugging, 104
- procedures, stored, invoking, 1214-1215
- processor types, displaying, 201
- profile property, CD:1563
- project object, 1312-1313
- prompt() method, 1147
- prompt property, CD:1601-1602
- properties, 1209-1210, 1246-1247, 1287, 1304, CD:1805
- protocol property, 589-590, 880, 893-894, 1332-1333, 1376
- prototype property, 361, 445-446, 1228, 1304, 1350-1351
- publicId property, reading, CD:1428, CD:1446, CD:1471
- QueryString collection, 1323
- Quit() method, CD:1937-1938
- Radio object, 939
- random() method, 427-428
- read() method, 1288, CD:1917-1918
- ReadAll() method, CD:1918-1919
- readByte() method, 1289
- ReadLine() method, CD:1919
- readln() method, 1290
- readOnly property, CD:1595, CD:1720-1721
- Rect object, reading properties, CD:1813
- red property, reading, CD:1817-1818
- redirect() function, browsers, 1316
- Redirect() method, 1343
- referrer property, 683, CD:1534
- RegDelete() method, CD:1997-1998
- RegExp object, 468
- RegExp.\$ property, 471
- RegExp.\$* property, 469-470
- RegExp.\$& property, 470
- RegExp.\$' property, 473
- RegExp.\$+ property, 474
- RegExp.\$1,\$2,...\$9 property, 474-475
- RegExp.\$` property, 472
- RegExp.valueOf() method, 491
- registerCFunction function, 1317
- registry entries, reading, CD:1998
- RegRead() method, CD:1998
- regular expressions, displaying, 254
- RegWrite() method, CD:1999
- rel property, CD:1486, CD:1613
- relatedTarget property, CD:1753-1754
- release() method, 1210-1211
- releaseEvents() method, 684-685, 852-853
- reload() method, 894-895
- remote scripting, 188-189

- remote scripts, CD:1953-1954, CD:1976
- Remove() method, 1382,
 - CD:1666-1667, CD:1959
- RemoveAll() method, 1382-1383
- removeAttribute() method, CD:1440
- removeAttributeNode() method,
 - CD:1441
- removeAttributeNS() method, CD:1441
- removeChild() method, CD:1469
- removeEventListener() method,
 - CD:1747
- removeMember() method, 404
- removeNamedItem() method, CD:1451
- removeNamedItemNS() method,
 - CD:1451-1452
- RemoveNetworkDrive() method,
 - CD:1969
- RemovePrinterConnection() method,
 - CD:1970
- removeProperty() method, CD:1798
- repetitive, reducing with the with statement, 75-76
- replace() method, 895
- replaceChild() method, CD:1470
- replicateData() method, CD:1411-1412
- Replyto property, 1364
- Request object, 1318, 1327-1328
- Request.ServerVariables collection, 177-178
- request.unwatch() method, 1333
- request.watch() method, 1334
- requesting agent properties, viewing, 1327-1328
- reset() method, 773, CD:1552-1553
- Reset object, 953
- RESET property, accessing, 740
- RESIZE property, 741
- resizeBy() method, 854, 1150-1151
- resizeTo() method, 855, 1152
- resultSet() method, 1394
- ResultSet object, 1345-1346
- resultsets, closing, 1346-1347
- return statement, 493
- returning values from functions, 79-80
- returnValue() method, 1395
- rev property, CD:1487, CD:1614
- RGBColor object, reading properties,
 - CD:1816
- right property, reading, CD:1814-1815
- rightContext property, 486
- rollbackTransaction() method, 1211-1212, 1247
- root folder, returning, CD:1839-1840
- round() method, 428-429
- routeEvent() method, 686-687, 855-856
- rowIndex property, CD:1708
- rows, 1226-1232
- rows property, CD:1562, CD:1698-1699,
 - CD:1713-1714, CD:1721
- rowSpan property, CD:1681-1682
- rules property, CD:1699
- Run() method, CD:2000
- runtime errors, catching, 99-100
- Save() method, CD:2009-2010,
 - CD:2021-2022
- scheme property, CD:1621
- scope property, CD:1682
- screen object, displaying properties, 967-968
- screenX property, 710, 1155, CD:1754
- screenY property, 711-712, 1155,
 - CD:1755
- script mode, retrieving, CD:1936
- ScriptEngine() function, 494
- ScriptEngineBuildVersion() function, 495
- ScriptEngineMajorVersion() function, 495
- ScriptEngineMinorVersion() function, 496
- ScriptFullName property, displaying,
 - CD:1938
- scripting engine version information, 113-114
- Scripting.Signer object, CD:1928
- ScriptName property, displaying,
 - CD:1938-1939
- ScriptTimeout property, 1371
- scroll() method, 1156-1157
- scrollbars.visible property, 1158
- scrollBy() method, 1159-1160
- scrolling property, CD:1559, CD:1575
- scrollTo() method, 1160-1161
- search property, 590-591, 880-881, 896
- sectionRowIndex property,
 - CD:1708-1709
- select box names, retrieving, 982-984
- select box properties, retrieving, 988-989
- select() method, 756-757, 930,
 - CD:1596, CD:1722
- text areas, selecting text, 1079-1080
- text boxes, selecting text, 1059

SELECT property, 742-743
SELECT queries, formatting, 1213, 1248-1249
selected property, 917, CD:1646
selected text, displaying, 659-660
selectedIndex property, 990-991, CD:1667
selectorText property, setting, CD:1779, CD:1800
self property, 788, 1161-1162
send() method, 1364-1365
SendKeys() method, CD:2002
SendMail object, 172-173, 1355-1357
separator property, reading, CD:1769
Server object, 1369
 accessing properties, 1373-1374
ServerVariables collection, 1326
Session object, 1379
SessionID property, 1383
Session_OnEnd event, 1384
SetAbort() method, 1311
setAttribute() method, CD:1442
setAttributeNode() method, CD:1442-1443
setAttributeNodeNS() method, CD:1443
setAttributeNS() method, CD:1444
SetComplete() method, 1311-1312
setDate() method (Date object), 308-309
SetDefaultPrinter() method, CD:1970-1971
setFloatValue() method, CD:1785-1786
setFullYear() method (Date object), 309-310
setHotKeys() method, 1162
setHours() method (Date object), 310
setInterval() method, 789, 1163
setMember() method, 405
setMilliseconds() method (Date object), 311
setMinutes() method (Date object), 312
setMonth() method (Date object), 312-313
setNamedItem() method, CD:1452
setNamedItemNS() method, CD:1453
setPosition() method, 1291
setProperty() method, CD:1798
setResizable() method, 1164
setSeconds() method (Date object), 313-314
setSlot() method, 406
setStringValue() method, CD:1786
setTime() method (Date object), 314
setTimeout() method, 789-790, 1164-1165
setUTCDate() method (Date object), 315
setUTCFullYear() method (Date object), 316
setUTCHours() method (Date object), 316-317
setUTCMilliseconds() method (Date object), 317
setUTCMinutes() method (Date object), 318
setUTCMonth() method (Date object), 319
setUTCSeconds() method (Date object), 319-320
setYear() method (Date object), 320-321
setZOptions() method, 1166
shape property, CD:1487, CD:1501
share names of drives, returning, CD:1841
shiftKey property, CD:1755
shortcuts, creating, 183
ShortName property, CD:1904
ShortPath property, CD:1905
ShowUsage() method, CD:1949-1950
siblingAbove property, 857
siblingBelow property, 858
sin() method, 429-430
size property, CD:1507, CD:1546, CD:1567, CD:1596-1597, CD:1668, CD:1906
Skip() method, CD:1920
SkipLine() method, CD:1921
Sleep() method, CD:1939
Smtpsrvr property, 1365
source property, 487, 1187, CD:1983-1984
SourceText property, CD:1984
span property, CD:1686-1687
SpecialFolders property, CD:2003
specified property, specifying attributes, CD:1405
splitText() method, CD:1474-1475
SQLTable() method, 1213, 1248-1249
sqrt() method, 430-431
SQRT2 property, 432
SQRT_2 property, 431
src property, 817, 858-859, CD:1560, CD:1576, CD:1583, CD:1597

- ssjs_generateClientID() function, 1386
- ssjs_getClientID() function, 1388
- standby property, CD:1633
- Start event, CD:1976-1977
- start property, CD:1638
- StaticObjects collection, 1385
- status property, 1166, 1344, CD:1977-1978, CD:1986
- statusbar.visible property, 1167-1168
- StdErr property, CD:1940, CD:1987
- StdIn property, CD:1940-1941, CD:1987-1988
- StdOut property, CD:1941, CD:1988-1989
- stop() method, 1168-1169
- stopPropagation() method, CD:1739
- stored procedures, 1250-1251, 1268-1269
- storedProc() method, 1214-1215, 1250
- storedProcArgs() method, 1251, 1268-1269
- Sproc object instances, creating, 1389
- string concatenations, + (addition operator), 53
- string data, retrieving, CD:1408
- String object, 500-501
- String() function, 497
- String() method, 379
- String.anchor() method, 501
- String.big() method, 502
- String.blink() method, 503
- String.bold() method, 504
- String.charAt() method, 504-505
- String.charCodeAtAt() method, 506
- String.concat() method, 507
- String.constructor property, 507-508
- String.fixed() method, 508
- String.fontcolor() method, 509
- String.fontSize() method, 510
- String.fromCharCode() method, 511
- String.indexOf() method, 511-512
- String.italics() method, 512-513
- String.lastIndexOf() method, 513-514
- String.length property, 514
- String.link() method, 515
- String.localeCompare() method, 516
- String.match() method, 516-517
- String.prototype property, 517-518
- String.replace() method, 519
- String.search() method, 520-521
- String.slice() method, 521
- String.small() method, 522
- String.split() method, 523-524
- String.strike() method, 524-525
- String.sub() method, 525
- String.substr() method, 526
- String.substring() method, 527-528
- String.sup() method, 528
- String.toLocaleLowerCase() method, 529
- String.toLocaleUpperCase() method, 529-530
- String.toLowerCase() method, 530-531
- String.toSource() method, 531
- String.toString() method, 532
- String.toUpperCase() method, 533
- String.unwatch() method, 533-534
- String.valueOf() method, 535
- String.watch() method, 535-536
- strings, 41, CD:1940-1941
- strings read, converting into bytes, 1291-1292
- stringToByte() method, 1291-1292
- style, overriding, CD:1809
- Style object, 997
- style property, CD:1771-1772, CD:1778-1779, CD:1800, CD:1811-1812
- style sheet classes, creating, 650
- style sheet IDs, creating, 663-664
- style sheet tags, creating, 689
- style sheets, creating, CD:1810
- styles, computing, CD:1818
- styleSheet property, reading, CD:1774
- SubFolders property, CD:1907
- Subject property, 1366
- Submit button, 1032-1034, 1037-1038
- Submit event, canceling, 94
- submit() method, 774, CD:1553
- Submit object, accessing values, 1043-1044
- SUBMIT property, 743-744
- substringData() method, CD:1409
- substrings, creating, CD:1409
- suffixes property, accessing, 902
- summary property, CD:1700
- sun object, 536-537
- sun package, 463-464
- switch statement, 538-539
- switch structures, 65
- system directory path, displaying, CD:1955-1957

- systemId property, reading, CD:1429, CD:1447, CD:1456, CD:1471-1472
- tabIndex property, CD:1488, CD:1516-1517, CD:1598, CD:1633-1634, CD:1668-1669, CD:1722
- tag names, CD:1422-142
- tagIndex property, CD:1502
- tagName property, reading, CD:1444-1445
- tags() method, 640-641
- taint() method, 1045
- taintEnabled() method, 912-913
- tan() method, 433
- target property, 592-593, 881-882, CD:1502-1505, CD:1614, CD:1739-1740
 - accessing, 712-713
 - reading, CD:1473
 - results, displaying, 775-776
- TargetPath property, CD:2010, CD:2022
- tBodies property, CD:1700
- Terminate() method, CD:1978, CD:1989
- test() method, 488
- text
 - area names, retrieving, 1071-1072
 - area values, 1066-1067, 1082-1083
 - colors, setting, 656
 - data length, checking, CD:1408-1409
 - displaying, 694-696
 - entering, CD:1940-1941
- text boxes
 - cursor, placing, 1050-1051
 - focus, removing, 1048
 - names, retrieving, 1054
 - text, selecting, 1059
 - values, 76, 1049, 1061-1062
- Text object instances, displaying properties, 1047
- text property, 882-883, 918, CD:1511, CD:1646-1647, CD:1655, CD:1725
- textAlign property, 1023
- Textarea object, displaying instance properties, 1064-1065
- textboxes, displaying contents, 1039-1040
- textDecoration property, 1024
- textIndent property, 1025
- TextStream object, CD:1913
- textTransform property, 1025-1026
- tFoot property, CD:1701
- tHead property, CD:1701-1702
- this keyword, 539-541
- throw keyword, 542-543
- time, converting, 321-322
- timed events, creating, 95-96
- Timeout property, 1385
- timeStamp property, CD:1740
- title property 690, CD:1535, CD:1542
- tmp folder, finding, CD:1876-1877
- To property, 1366
- toExponential() method, 446
- toFixed() method, 447
- toLocaleString() method, 447-448
- toolbar.visible property, 1169-1170
- top property, 790, 859-860, 1170-1171, CD:1815
- toPrecision() method, 448
- toSource() method, 276-277, 362, 434, 449, 455-456, 488-489
- toString() method, 277-278, 324, 406, 434-435, 1215-1216, 1252-1253
- TotalBytes property, 1326-1327
- toUTCString() method (Date object), 324-325
- Trace Information window, writing information to, 1273
- transactions
 - rolling back, 1211-1212, 1247
 - starting, 1201-1203, 1236-1237
- Transfer() method, 1372
- Try...catch...finally statement, 545-546
- type conversions, == equal comparison operator, 56
- type property, CD:1489, CD:1517, CD:1608, CD:1804, CD:1907-1908
 - accessing, 713-714, 902-903
 - alert boxes, code to display, 1059-1060
 - buttons, accessing type, 609-610
 - displaying, 992, 1080-1081
 - rule type reading, CD:1789
- typeof unary operator, 547
- undefined property, 381, 548
- unescape() method, 380, 549
- Unload events, 680
- unlock() method, 1305, 1314, 1377
- unlocking, 1314
- unnamed objects, displaying, CD:2016-2017

- unnamed parameters, displaying, CD:1951, CD:2017-2018
- Unnamed property, displaying unnamed parameters, CD:1951
- untaint() method, 1084
- unwatch() method, 450, 611, 758-759, 805-806
 - Area object, 594-595
 - Array object, 270
 - Boolean object, 278
 - Date object, 325
- updateRow() method, 1231-1232
- URL property, 692, CD:1535-1536
- URLEncode() method, 1372
- URLs, displaying, 670-671
- useMap property, CD:1583-1584, CD:1599, CD:1635
- user interface events, CD:1761-1763
- userAgent property, 913
- UserDomain property, CD:1971
- UserName property, CD:1972
- vAlign property, CD:1683, CD:1687, CD:1709, CD:1714
- value property, 797, 920-921, CD:1599-1600, CD:1647, CD:1723-1724
 - accessing, 990-991
 - attributes, writing values, CD:1405-1406
 - buttons, modifying text, 612-613
 - Submit object values, accessing, 1043-1044
 - text area values, resetting, 1082-1083
 - text boxes, resetting values, 1061-1062
- valueOf() method, 271, 279, 327, 363-364, 458
- valueType property, CD:1652
- var keyword, 549-550
- variable scope, 38-39
- variables, declaring and assigning, 37-38
- variables (custom compiled), creating, 202
- VBAArray object, 551-552
- VBAArray.dimensions() method, 552-553
- VBAArray.getItem() method, 554
- VBAArray.lbound() method, 555-556
- VBAArray.toArray() method, 556-557
- VBAArray.ubound() method, 558
- version property, CD:1569, CD:1942
- visibility property, 862
- visible property, 1123-1125, 1145-1146, 1158, 1167-1170
- vLink property, CD:1511-1512
- void operator, 559
- vspace property, CD:1635
- vspace property, 819, CD:1496, CD:1584
- watch() method, 328, 451-452, 613-614, 806-807, 921-922, 1044-1045
 - Area object, 596-597
 - Array object, 272
 - Boolean object, 279-280
- Web browser layers, manipulating, 153-154
- which property, accessing, 718
- while statement, 560-561
- whiteSpace property, 1028
- width property, 719, 820-821, CD:1497, CD:1565, CD:1636
- win16 processors, alerting when on, 198
- win32 system, alerting when on, 199
- window focus, removing, 1092-1093
- Window object instances, creating, 1088-1090
- window property, 792-793, 1173
- window.captureEvents() method, 1095-1096
- window.document property methods, 1109-1110
- window.releaseEvents() method, 1149
- window.routeEvent() method, 1153-1155
- window.scrollbars property, 1157
- window.statusbar property, 1167
- window.toolbar property, 1169
- WindowStyle property, CD:2011 with statement, 561-562
- WorkingDirectory property, CD:2012
- write() function, 1398-1399
- write() method, 694-695, 1295-1296, 1335, CD:1536, CD:1921-1922
- WriteBlankLines() method, CD:1922
- writeByte() method, 1296-1297
- WriteLine() method, CD:1923
- writeln() method, 695-696, 1297-1298, CD:1537
- WScript object, CD:1929-1930
- WScript parameters, displaying, CD:1946-1947
- WSH (Windows Script Host) file, 180, CD:1936-1937

- WshArguments object, CD:1945
- WshController object, creating, CD:1952-1953
- WshEnvironment object, displaying system directory path, CD:1955-1956
- WshNamed object, CD:1960
- WshNamed.Exists() method, CD:1962
- WshNetwork object, CD:1933, CD:1964
- WshRemote object, CD:1973
- WshRemoteError object, creating, CD:1979
- WshScriptExec object, creating, CD:1985
- WshShell object, creating, CD:1990-1991
- WshShortcut object, creating, CD:2005
- WshSpecialFolders object, creating, CD:2013
- WshUnnamed object, CD:2016-2017
- WshUrlShortcut object, CD:2020
- XML documents, adding comments, CD:1416
- XUL (XML-based User Interface Language), 155-156
- zIndex property, 864
- code property, CD:1627**
 - DOMException object, CD:1430
 - EventExcpetion object, CD:1741
 - HTMLAppletElement object, CD:1490
 - HTMLObjectElement object, CD:1624
- codeBase property, CD:1490, CD:1493, CD:1624, CD:1627-1628**
- codeConnection.prototype property, 1209-1210**
- CodePage property, 1379-1380**
- codeType property (HTMLObjectElement object), CD:1624, CD:1628**
- collections**
 - Application object, 1177
 - Application.Contents, 1179-1180
 - Application.StaticObjects, 1182
 - ClientCertificate (Request object), 1318-1320
 - Contents, 1177, 1379-1379
 - Cookies, 1318, 1321, 1335, 1340
 - document.all, 147
 - Drives, CD:1842-1845, CD:1865, CD:1874
 - Folders, CD:1908-1909
 - Form (Request object), 1318, 1322
 - QueryString (Request object), 1318, 1323
 - Request object, 1318
 - Request.ClientCertificate, 1319
 - Request.Cookies, 1321
 - Request.Form, 177, 1321
 - Request.QueryString, 1323
 - Request.ServerVariables, 177-178, 1324
 - Response object, 1335
 - Response.Cookies, 1339-1340
 - ServerVariables, 1318, 1324-1326
 - Session object, 1379
 - Session.Contents, 1381
 - Session.StaticObjects, 1385
 - StaticObjects, 1177, 1379, 1385
- colon (:), 72, 74, 284, 573, 580, 589, CD:2016**
- color property**
 - code, 1007, CD:1506, CD:1545
 - HTMLBaseFontElement object, CD:1505
 - HTMLFontElement object, CD:1544
 - Style object, 649, 662, 688, 995
- colorDepth property, 971**
- colors**
 - backgrounds, code to set, 353
 - of links, 669-670, 692-693
 - parameter, displaying, CD:1948-1949
 - text, code to set, 656
- cols property, CD: 1560-1561, CD:1715-1717**
- colSpan property (HTMLTableCellElement object), CD:1674, CD:1679**
- column names (tables), code, 1223-1224**
- Column property, code, 1183-1185, CD:1913, CD:1916**
- columnName() method, 1220-1224, 1345-1348**
- columnName property (Cursor object), 1219, 1222**
- columns() method, 1220, 1224, 1345, 1348-1349**
- COM (Component Object Model) objects, CD:1824**
- COM servers, CD:1824**
- comma (,), 214-215**
- command lines**
 - parameters, 160-161, CD:1931
 - scripts, executing syntax, 185
 - switches, CD:1936

commands

- Break at Next Statement, 103
- Edit menu, Preferences, 136
- WSH (Windows Script Host) scripts, 186-187

COMMENT NODE constant (Node object), CD:1454

Comment object, CD:1412

comments, 17, 33, 216-217, CD:1416

commitFlag parameter, 171, 1238, 1256, 1259, 1262

commitTransaction() method, 1199, 1202-1203, 1234, 1237

common gateway interfaces (CGIs), 5

compact property, CD: 1521-1525, CD:1617-1618, CD:1636-1637, CD:1726

comparing

- ?: conditional operator and if statement, 59, 218-219
- JavaScript and Java, 13
- JavaScript and JScript, 6-7, CD:1824
- pre- and post-decrements, 205, 227
- prevValue property and newValue property, code, CD:1760-1761
- server-side and client-side, 158-159

comparison operators, 55-58

compilations, conditional, code, 200

compile() method (RegExp object), 467, 476-477

compiled variables (custom), code to create, 202

compilers, jsac (JavaScript Application Compiler), 160

compiling scripts, 160-161

complete property (Image object), 807, 810

Component Object Model (COM) objects, CD:1824

ComputerName property (WshNetwork object), CD:1964-1967

COMSPEC (Windows environment variable), CD:1955

concat() method, 42, 251-252, 498

concatenating numbers, code, 225

concatenations, strings, + (addition operator), 53

conditional compilations, code, 200

conditional operator (?), 58-59, 218

conditional statements, 62-65

configurations, Server Side JavaScript Application Manager, 162-165

confirm boxes, code, 1103

confirm() method (Window object), 1085 1103

Confirm On option (Server Side JavaScript Application Manager), 165

connect() method, 1234, 1238-1239, 1257-1264

connected() method, 1199, 1203-1204, 1234, 1239-1240, 1257, 1260

connecting to databases, code, 1238-1239, 1259-1260

connection errors (databases), code to retrieve, 1243-1245, 1264-1267

connection() method (DbPool object), 1257, 1261

Connection object, 1199-1200

Connection.beginTransaction() method, 1201

Connection.commitTransaction() method, 1202

Connection.connected() method, 1203

Connection.connection() method, 1234

Connection.cursor() method, 1204

Connection.execute() method, 1205

Connection.majorErrorCode() method, 1205

Connection.majorErrorMessage() method, 1206

Connection.minorErrorCode() method, 1207

Connection.minorErrorMessage() method, 1208

Connection.prototype property, 1209-1210

Connection.release() method, 1210

Connection.rollbackTransaction() method, 1211

Connection.SQLTable() method, 1212

Connection.storedProc() method, 1213-1214

Connection.toString() method, 1215

Connection.unwatch() method, 1216

Connection.watch() method, 1218

connections

- databases, 170-172, 1210, 1260
- errors, code to retrieve, 1205-1208
- pools, code, 171
- releasing to pools, code, 1210-1211
- testing, code, 1203-1204, 1239-1240

ConnectObject() method (WScript object), CD:1929:1932

const keyword, 283

constants

ADDITION (MutationEvent object),
CD:1756
AT TARGET (Event object), CD:1733
ATTRIBUTE NODE (Node object),
CD:1454
BUBBLING PHASE (Event object),
CD:1733
CAPTURING PHASE (Event object),
CD:1733
CDATA SECTION NODE (Node
object), CD:1454
COMMENT NODE (Node object),
CD:1454
CSS ATTR (CSSPrimitiveValue object),
CD:1780
CSS CM (CSSPrimitiveValue object),
CD:1780
CSS COUNTER (CSSPrimitiveValue
object), CD:1780
CSS CUSTOM (CSSValue object),
CD:1805
CSS DEG (CSSPrimitiveValue object),
CD:1780
CSS DIMENSION (CSSPrimitiveValue
object), CD:1780
CSS EMS (CSSPrimitiveValue object),
CD:1780
CSS EXS (CSSPrimitiveValue object),
CD:1780
CSS GRAD (CSSPrimitiveValue object),
CD:1780
CSS HZ (CSSPrimitiveValue object),
CD:1780
CSS IDENT (CSSPrimitiveValue
object), CD:1780
CSS IN (CSSPrimitiveValue object),
CD:1780
CSS INHERIT (CSSValue object),
CD:1805
CSS KHZ (CSSPrimitiveValue object),
CD:1780
CSS MM (CSSPrimitiveValue object),
CD:1780
CSS MS (CSSPrimitiveValue object),
CD:1780
CSS NUMBER (CSSPrimitiveValue
object), CD:1780
CSS PC (CSSPrimitiveValue object),
CD:1780
CSS PERCENTAGE
(CSSPrimitiveValue object), CD:1780

CSS PRIMITIVE VALUE (CSSValue
object), CD:1805
CSS PT (CSSPrimitiveValue object),
CD:1780
CSS PX (CSSPrimitiveValue object),
CD:1781
CSS RAD (CSSPrimitiveValue object),
CD:1781
CSS RECT (CSSPrimitiveValue object),
CD:1781
CSS RGBCOLOR (CSSPrimitiveValue
object), CD:1781
CSS S (CSSPrimitiveValue object),
CD:1781
CSS STRING (CSSPrimitiveValue
object), CD:1781
CSS UNKNOWN (CSSPrimitiveValue
object), CD:1781
CSS URI (CSSPrimitiveValue object),
CD:1781
CSS VALUE LIST (CSSValue object),
CD:1805
CSSPrimitiveValue object, CD:1780
CSSValue object, CD:1805
DOCUMENT FRAGMENT NODE
(Node object), CD:1454
DOCUMENT NODE (Node object),
CD:1454
DOCUMENT TYPE NODE (Node
object), CD:1454
DOMSTRING SIZE ERR
(DOMException object), CD:1429
ELEMENT NODE (Node object),
CD:1454
ENTITY NODE (Node object),
CD:1454
ENTITY REFERENCE NODE (Node
object), CD:1454
Event object, CD:1733
EventException object, CD:1741
HIERARCHY REQUEST ERR
(DOMException object), CD:1429
INDEX SIZE ERR (DOMException
object), CD:1429
INUSE ATTRIBUTE ERR
(DOMException object), CD:1430
INVALID ACCESS ERR
(DOMException object), CD:1430
INVALID CHARACTER ERR
(DOMException object), CD:1429
INVALID MODIFICATION ERR
(DOMException object), CD:1430

- INVALID STATE ERR (DOMException object), CD:1430
- Math.E, 34
- Math.LN2, 34
- Math.LN10, 34
- Math.LOG2E, 34
- Math.LOG10E, 35
- Math.PI, 35
- Math.SQRT1_2, 35
- Math.SQRT2, 35
- MODIFICATION (MutationEvent object), CD:1756
- MutationEvent object, CD:1756
- NAMESPACE ERR (DOMException object), CD:1430
- NO DATA ALLOWED ERR (DOMException object), CD:1429
- NO MODIFICATION ALLOWED ERR (DOMException object), CD:1429
- Node object, CD:1454
- NOT FOUND ERR (DOMException object), CD:1429
- NOT SUPPORTED ERR (DOMException object), CD:1430
- NOTATION NODE (Node object), CD:1454
- Number.MAX_VALUE, 35
- Number.MIN_VALUE, 35
- Number.NaN, 35
- Number.NEGATIVE_INFINITY, 35
- Number.POSITIVE_INFINITY, 35
- numerical, 34-35
- PROCESSING INSTRUCTION NODE (Node object), CD:1454
- REMOVAL (MutationEvent object), CD:1756
- SYNTAX ERR (DOMException object), CD:1430
- TEXT NODE (Node object), CD:1454
- UNSPECIFIED EVENT TYPE ERR (EventExcpetion object), CD:1741
- WRONG DOCUMENT ERR (DOMException object), CD:1429
- constructor argument (Date object), 286**
- constructor property**
 - code, 359, 475-476, 1280, 1301-1302, 1359
 - Array object, 249, 253
 - Boolean object, 273-275
 - Date object, 289
 - Function object, 353
 - Number object, 439-441
 - Object object, 452-454
 - RegExp object, 467
- constructors**
 - Function(), 352
 - Number, code, 440
 - Object(), creating arrays, 44-45
- contatenating strings, 50**
- CONTENT LENGTH variable (ServerVariables collection), 1325**
- content property (HTMLMetaElement object), CD:1618-1619**
- CONTENT TYPE variable (ServerVariables collection), 1325**
- contentDocument property**
 - code, CD:1629
 - HTMLFrameElement object, CD:1554
 - HTMLIFrameElement object, CD:1570
 - HTMLObjectElement object, CD:1624
 - reading, code, CD:1555, CD:1571
- Contents collection, 1177, 1379-1381**
- Contents.Lock() method (Application object), 1178**
- Contents.Remove() method, 1178, 1379**
- Contents.RemoveAll() method, 1178, 1379**
- Contents.Unlock() method (Application object), 1178**
- ContentType property, code, 1339**
- contextual() method, 637, 652**
- continue keyword, 73-74, 283**
- continue statement, 73-75, 283-285**
- controls, built-in for Web browsers, 116-120**
- ConversionError (runtime exception error, 344**
- conversions, type, 39, 56-57**
- converting**
 - arrays, Visual Basic to JScript, CD:1829
 - bytes into strings, code, 1277-1278
 - dates, code, 321-322
 - hexadecimal values to ASCII equivalents, 380
 - strings read into bytes, code, 1291-1292
 - time, code, 321-322
- cookie property, 636, 653, 1109, CD:1525, CD:1529**
- cookies, 136-139, 653**
- Cookies collection 1318, 1321, 1340, 1335**
- cookies.txt file, 136**
- coords property, CD:1480, CD:1483, CD:1497-1499**

Copy() method, CD:1845-1848, CD:1891, CD:1894

CopyFile() method (FileSystemObject object), CD:1864-1867

CopyFolder() method (FileSystemObject object), CD:1864, CD:1867-1868

copying files, code, CD:1848, CD:1867

core ECMAScript object hierarchy, 10

core elements of programming, 31

core language, 193

core Microsoft or Netscape object hierarchy, 11

core objects, 10-11

cos() method, 408, 415-416

Count() method

arguments, code to retrieve, CD:1961

code, CD:2014

displaying, code, CD:1946

environment variables, code to display, CD:1956

unnamed parameters, code to display, CD:2017-2018

WshArguments object, CD:1944

WshEnvironment object, CD:1954

WshNamed object, CD:1960

WshSpecialFolders object, CD:2013

WshUnnamed object, CD:2016

Count property, CD:1945

code, 1322, 1324, CD:1910-1911

Dictionary object, CD:1824-1826

Drives collection, CD:1843-1844

Files object, CD:1861-1863

Folders collection, CD:1909

WshArguments object, CD:1944

Counter object, CD:1767-1768

counter values, code, CD:1782

Counter.identifier property, CD:1768

Counter.listStyle property, CD:1768-1769

Counter.separator property, CD:1768-1769

create parameter (OpenTextFile() method), CD:1890

createAttribute() method, CD:1413-1414

createAttributeNS() method (Document object), CD:1413

createCaption() method, CD:1689, CD:1693

createCDATASection() method (Document object), CD:1413

createComment() method (Document object), CD:1413

createCSSStyleSheet() method (DOMImplementationCSS object), CD:1810-1811

createDocument() method (DOMImplementation object), CD:1431-1432

createDocumentFragment() method, CD:1413, CD:1416-1417, CD:1425

createDocumentType() method (DOMImplementation object), CD:1431-1433

createElement() method, 149, CD:1413

createElementNS() method (Document object), CD:1413

createEntityReference() method (Document object), CD:1413

createEvent() method, CD:1731-1734

CreateFolder() method (FileSystemObject object), CD:1864, CD:1869

createHTMLDocument() method (HTMLDOMImplementation object), CD:1537-1538

CreateObject() method, 1368-1369 object, CD:1929, CD:1933, CD:1953-1954

createProcessingInstruction() method (Document object), CD:1413

CreateScript() method, CD:1953

CreateShortcut() method (WshShell object), CD:1990-1992

CreateTextfile() method (FileSystemObject object), CD:1864, CD:1869-1870

createTextNode() method (Document object), CD:1413, CD:1420

createTFoot() method (HTMLTableElement object), CD:1689, CD:1694

createTHead() method (HTMLTableElement object), CD:1689, CD:1694

cross browser DHTML, 151-154

crypto property, 1103-1104, 1087

crypto.random() method, 1085, 1104

crypto.signText method (Window object), 1085, 1105

Cscript.exe command line switches, CD:1936

cscript.exe file, options, 185

CSS (cascading style sheets), 149

CSS ATTR constant (CSSPrimitiveValue object), CD:1780

- CSS CM constant (CSSPrimitiveValue object), CD:1780
- CSS COUNTER constant (CSSPrimitiveValue object), CD:1780
- CSS CUSTOM constant (CSSValue object), CD:1805
- CSS DEG constant (CSSPrimitiveValue object), CD:1780
- CSS DIMENSION constant (CSSPrimitiveValue object), CD:1780
- CSS EMS constant (CSSPrimitiveValue object), CD:1780
- CSS EXS constant (CSSPrimitiveValue object), CD:1780
- CSS GRAD constant (CSSPrimitiveValue object), CD:1780
- CSS HZ constant (CSSPrimitiveValue object), CD:1780
- CSS IDENT constant (CSSPrimitiveValue object), CD:1780
- CSS IN constant (CSSPrimitiveValue object), CD:1780
- CSS INHERIT constant (CSSValue object), CD:1805
- CSS KHZ constant (CSSPrimitiveValue object), CD:1780
- CSS MM constant (CSSPrimitiveValue object), CD:1780
- CSS MS constant (CSSPrimitiveValue object), CD:1780
- CSS NUMBER constant (CSSPrimitiveValue object), CD:1780
- CSS PC constant (CSSPrimitiveValue object), CD:1780
- CSS PERCENTAGE constant (CSSPrimitiveValue object), CD:1780
- CSS PRIMITIVE VALUE constant (CSSValue object), CD:1805
- CSS PT constant (CSSPrimitiveValue object), CD:1780
- CSS PX constant (CSSPrimitiveValue object), CD:1781
- CSS RAD constant (CSSPrimitiveValue object), CD:1781
- CSS RECT constant (CSSPrimitiveValue object), CD:1781
- CSS RGBCOLOR constant (CSSPrimitiveValue object), CD:1781
- CSS S constant (CSSPrimitiveValue object), CD:1781
- CSS STRING constant (CSSPrimitiveValue object), CD:1781
- CSS UNKNOWN constant (CSSPrimitiveValue object), CD:1781
- CSS URI constant (CSSPrimitiveValue object), CD:1781
- CSS VALUE LIST constant (CSSValue object), CD:1805
- CSSCharsetRule object, CD:1770
- CSSCharsetRule.encoding property, CD:1770
- CSSFontFaceRule object, CD:1771
- CSSFontFaceRule.style property, CD:1771-1772
- CSSImportRule object, CD:1772
- CSSImportRule.href property, CD:1773
- CSSImportRule.media property, CD:1773
- CSSImportRule.styleSheet property, CD:1774
- CSSMediaRule object, CD:1774-1775
- CSSMediaRule.cssRules property, CD:1775
- CSSMediaRule.deleteRule() method, CD:1776
- CSSMediaRule.insertRule() method, CD:1776-1777
- CSSMediaRule.media property, CD:1777
- CSSPageRule object, CD:1778
- CSSPageRule.selectorText property, CD:1778
- CSSPageRule.style property, CD:1779
- CSSPrimitiveValue object, CD:1779-1781
- CSSPrimitiveValue.getCounterValue() method, CD:1782
- CSSPrimitiveValue.getFloatValue() method, CD:1782
- CSSPrimitiveValue.getRectValue() method, CD:1783
- CSSPrimitiveValue.getRGBColorValue() method, CD:1783-1784
- CSSPrimitiveValue.getStringValue() method, CD:1784
- CSSPrimitiveValue.primitiveType property, CD:1785
- CSSPrimitiveValue.setFloatValue() method, CD:1785
- CSSPrimitiveValue.setStringValue() method, CD:1786
- CSSRule object, CD:1786-1787
- cssRule property (CSSStyleSheet object), CD:1801

CSSRule.cssText property, CD:1787-1788
CSSRule.parentRule property, CD:1788
CSSRule.parentStyleSheet property, CD:1789
CSSRule.type property, CD:1789
CSSRuleList object, CD:1790, CD:1794, CD:1806, CD:1817
CSSRuleList.item() method, CD:1791
CSSRuleList.length property, CD:1791
cssRules property (CSSMediaRule object), CD:1775-1776, CD:1801-1802
CSSStyleDeclaration object, CD:1792-1793
CSSStyleDeclaration.cssText property, CD:1793
CSSStyleDeclaration.getPropertyCSSValue() method, CD:1794
CSSStyleDeclaration.getPropertyPriority() method, CD:1794-1795
CSSStyleDeclaration.getPropertyValue() method, CD:1795
CSSStyleDeclaration.item() method, CD:1796
CSSStyleDeclaration.length property, CD:1796
CSSStyleDeclaration.parentRule property, CD:1797
CSSStyleDeclaration.removeProperty() method, CD:1797
CSSStyleDeclaration.setProperty() method, CD:1798
CSSStyleRule object, CD:1799
CSSStyleRule.selectorText property, CD:1799
CSSStyleRule.style property, CD:1800
CSSStyleSheet interface, 149
CSSStyleSheet object, CD:1800-1801
CSSStyleSheet.cssRules property, CD:1801
CSSStyleSheet.deleteRule() method, CD:1802
CSSStyleSheet.insertRule() method, CD:1802-1803
CSSStyleSheet.ownerRule property, CD:1803
cssText property, CD:1787-1788, CD:1792-1794, CD:1805-1806
CSSUnknownRule object, CD:1804
CSSValue object, CD:1804-1805
CSSValue.cssText property, CD:1805
CSSValue.cssValueType property, CD:1806
CSSValueList object, CD:1806-1807
CSSValueList.item() method, CD:1807

CSSValueList.length property, CD:1808
cssValueType property (CSSValue object), CD:1805-1806
ctrlKey property (MouseEvent object), CD:1748-1752
curly brackets ({}), 63-65, 77
current property, 799-801, 1117
CurrentDirectory property (WshShell object), CD:1990-1993
currentTarget property (Event object), CD:1733, CD:1736
Cursor
 columnName property, 1222
 object, 1219-1220
 prototype property, 1228
cursor() method, 1199, 1204, 1234, 1240
Cursor.close() method, 1221
Cursor.columnName() method, 1222-1223
Cursor.columnName property, 1222
Cursor.columns() method, 1224
Cursor.deleteRow() method, 1225
Cursor.insertRow() method, 1226
Cursor.next() method, 1227
Cursor.prototype property, 1228
Cursor.unwatch() method, 1229
Cursor.updateRow() method, 1231
Cursor.watch() method, 1232
cursors
 closing, code, 1221-1222, 1389-1390
 in columns, code, 1224
 in rows, code, 1226-1232
 in select boxes, code to place, 978-979
 in text areas, code to place, 1067-1068
 in text boxes, code to place, 1050-1051
custom compiled variables, code to create, 202

D

-d command-line parameter, 161
D drive, code to find, CD:1873
dailyTask() function, 95
data, 20, 1091-1094, CD:1410-1412
data property, 698, CD:1407-1408, CD:1472-1473, CD:1624
data tainting, 20-21
data types, 34-36
database object, 13, 1233-1234
database.beginTransaction() method, 1235
database.commitTransaction() method, 1236

database.connect() method, 1237-1238
database.connected() method, 1239
database.cursor() method, 1240
database.disconnect() method, 1241
database.execute() method, 1242
database.majorErrorCode() method, 1242-1243
database.majorErrorMessage() method, 1243
database.minorErrorCode() method, 1244
database.minorErrorMessage() method, 1245
database.prototype property, 1246-1247
database.rollbackTransaction() method, 1247
database.SQLTable() method, 1248
database.storedProc() method, 1249
database.storedProcArgs() method, 1250
database.toString() method, 1251-1252
database.unwatch() method, 1253
database.watch() method, 1254
Database/Function error, 167
databases
 access, user sessions, 171
 BLOB (binary large objects), 1188
 connecting to, code, 1238-1239, 1259-1260
 connection errors, code to retrieve, 1205-1208, 1243-1245, 1264-1267
 connections, code, 1241-1242, 1260
 connectivity, 170-172
 Database/Function error, 167
 disconnecting from, code, 1263-1264
 DML queries, code to run, 1205
 errors, 167
 LiveWire Services error, 167
 queries, code to run, 1204, 1240
 SELECT queries, code to format, 1213
 Standard Server-Side JavaScript error, 167
 stored procedures, 1214-1215, 1250-1251
 Vendor error, 167
datatypes, 35-36
Date object, 285-291, 305-328
DateCreated property, CD:1845, CD:1848-1849, CD:1891, CD:1894-1895
DateLastAccessed property, CD:1845, CD:1849-1850, CD:1891, CD:1895-1896
DateLastModified property, CD:1845, CD:1850-1851, CD:1891, CD:1896-1897

dates
 converging, code, 321-322
 creating, code, 326-327
 files, dating, CD:1848-1851
 modularizing, 26
 in universal time, code to access, 324-325
 verifying, 17
dateTime property (HTMLModElement object), CD:1622-1623
day argument, 286, 326
dbInstance
 parameter, 171, 1238, 1256-1258, 1262
 return value (toString() method), 1215, 1252, 1269
DbClick event, 90
DBCLICK
 event (Event object), 698
 property, code to access, 724-725
DbClick events, code, 673
DBName parameter, 171
dbName
 parameter, 1238, 1256, 1259, 1262
 return value (toString() method), 1215, 1252, 1269
DbPool() method, 1257, 1262-1263
DbPool object, 13, 170, 1255-1258, 1262-1263
DbPool.connect() method, 1258
DbPool.connected() method, 1260
DbPool.connection() method, 1199, 1261
DbPool.DbPool() method, 1262
DbPool.disconnect() method, 1263
DbPool.majorErrorCode() method, 1264
DbPool.majorErrorMessage() method, 1264-1265
DbPool.minorError Message() method, 1266
DbPool.minorErrorCode() method, 1265
DbPool.prototype property, 1267-1268
DbPool.storedProcArgs() method, 1268
DbPool.toString() method, 1269
DbPool.unwatch() method, 1270
DbPool.watch() method, 1272
DBType parameter, 171
dbType
 parameter, 1238, 1256-1258, 1262
 return value (toString() method), 1215, 1252, 1269
Debug control option (Server Side JavaScript Application Manager), 166

Debug Output option (Server Side**JavaScript Application Manager), 165****debug() function, 1273****debug() method, 167****debugger keyword, 328****debugging, 101-106****Debugging window, 166****DEC Alpha processor, 194****decimal integers, 34****declare property (HTMLElement object), CD:1624, CD:1630****declaring variables, 37-38****decodeURI()**

function, 329, 367

method (Global object), 365

decodeURIComponent()

function, 330-331, 368

method (Global object), 365

decoding data, 1091, 1094**decrement operator (--), 204-205****decrements, pre- and post-, 205, 227****default application settings (Server Side JavaScript Application Manager), 164-165****default data tainting, 21****default keyword, 331****Default Page application setting (Server Side JavaScript Application Manager), 164****defaultChecked property**

check boxes, code to reset, 621-622

Checkbox object, 615

code, 942-943, CD:1591

HTMLInputElement object, CD:1586

Radio object, 938

defaultSelected property, 915-916, CD:1641-1642**defaultStatus property, 1087, 1106****defaultValue property**

code, 924-925, 1049, CD:1592,

CD:1717-1718

HTMLInputElement object, CD:1586

HTMLTextAreaElement object,

CD:1715

Password object, 922

text areas, values, code to set, 1066-1067

Text object, 1046

Textarea object, 1064

defer property (HTMLScriptElement object), CD:1655-1656**defining**

Car object, code, 404-405

event handlers with dot notation, code, 93

patterns, 86-88

RegExp objects, 86

definitions

language independent, 27

true, 55

delete argument (Array object), 266**Delete() method, CD:1845, CD:1851-1852, CD:1891, CD:1897****delete operator, 331-332****deleteCaption() method**

(HTMLTableElement object), CD:1689, CD:1695

deleteCell() method

(HTMLTableRowElement object), CD:1703, CD:1707

deleteData() method (CharacterData object), CD:1407, CD:1411**DeleteFile() method (FileSystemObject object), CD:1864, CD:1871****DeleteFolder() method (FileSystemObject object), CD:1864, CD:1872****deleteResponseHeader() function, 1274****deleteRow() method**

code, 1225-1226, CD:1695-1696, CD:1712

Cursor object, 1220

HTMLTableElement object, CD:1689

HTMLTableSectionElement object, CD:1710

deleteTFoot() method

(HTMLTableElement object), CD:1689, CD:1696

deleteTHead() method

(HTMLTableElement object), CD:1689, CD:1697

deleting

content.type of files, code, 1274

environment variables, code, CD:1959

files, code, CD:1851-1852, CD:1871

folders, code, CD:1872

dependent parameter (Window object), 1134**description argument (Error object), 341****Description property, 1183, 1186, CD:1979-1981, CD:2005-2007****description property**

accessing, code, 900

code, 935

- Error object, 341
- MimeType object, 899
- Plugin object, 934
- destination parameter, CD:1867-1868**
- destroy() method (client object), 1193-1195**
- destroying client object properties, code, 1195**
- detail property (UIEvent object), CD:1761-1762**
- DevEdge Web site, 111**
- development history of programming languages, 5-7**
- DHTML (Dynamic HTML), 4, 145-148, 151-154**
- dialog boxes, Java Security, 22**
- dictionaries, CD:1825-1832**
- Dictionary object, CD:1824-1833**
- dimensions() method (VBArray object), 551**
- dir property (HTMLElement object), CD:1539**
- directories**
 - directories parameter (Window object), 1134
 - files, code to move to, CD:1853
 - \\netpub\\ASPSamp\\Samples, 1320
 - system directory path, code to display, CD:1955-1957
 - WSH, code to retrieve, CD:1937
- disabled property**
 - code, CD:1515, CD:1592-1593, CD:1611, CD:1640, CD:1662, CD:1718
 - HTMLButtonElement object, CD:1513
 - HTMLInputElement object, CD:1586
 - HTMMLinkElement object, CD:1609
 - HTMLOptGroupElement object, CD:1639
 - HTMLOptionElement object, CD:1641
 - HTMLSelectElement object, CD:1659
 - HTMLStyleElement object, CD:1670
 - HTMLTextAreaElement object, CD:1715
- disableExternalCapture() method, 1106-1107, 1085**
- disconnect() method, 1234, 1241-1242, 1257**
- disconnecting database connections, code, 1241-1242, 1263-1264**
- DisconnectObject() method (WScript object), CD:1929**
- dispatchEvent() method (EventTarget object), CD:1744-1746**
- display property, 649, 662, 688, 995, 1008**
- displayArguments() function, 79**
- <div> tag, 147, 151**
- division assignment operator (/=), 217**
- division operator (/), 47, 215**
- DML queries, 1205, 1242**
- do keyword, 332**
- docType property, CD:1412, CD:1420-1421**
- DOCUMENT FRAGMENT NODE constant (Node object), CD:1454**
- DOCUMENT NODE constant (Node object), CD:1454**
- document object, 19-21**
- Document object, 636-637, CD:1412-1413, CD:1425, CD:1435**
- Document Object Model (DOM), 12, 146-151**
- document property, 778-784, 822, 834-835, 1087**
- DOCUMENT TYPE NODE constant (Node object), CD:1454**
- document.alinkColor property, 637-638**
- document.all collection, 147**
- document.all property, 638**
- document.all property array, 639**
- document.all.item() method, 639**
- document.all.tags() method, 640**
- document.anchors property, 641**
- document.anchors.length property, 642**
- document.applets property, 643**
- document.applets.length property, 644**
- document.bgColor property, 645**
- document.captureEvents() method, 646**
- document.classes property, 648**
- document.close() method, 651**
- document.contextual() method, 652**
- document.cookie property, 652-653**
- Document.createAttribute() method, CD:1414**
- Document.createAttributeNS() method, CD:1414**
- Document.createCDATASection() method, CD:1415**
- Document.createComment() method, CD:1416**
- Document.createElement() method, CD:1417**
- Document.createElementNS() method, CD:1417-1418**

Document.createEntityReference()
method, CD:1418

Document.createProcessingInstruction()
method, CD:1419

Document.createTextNode() method,
CD:1419-1420

Document.docType property, CD:1420

Document.documentElement property,
CD:1421

document.domain property, 653

document.embeds property, 654

document.embeds.length property, 655

document.fgColor property, 655

document.formName property, 656

document.forms property, 657

document.forms.length property, 658

Document.getElementById() method,
CD:1421

Document.getElementsByTagName()
method, CD:1422

Document.getElementsByTagNameNS()
method, CD:1422-1423

document.getSelection() method, 659

document.handleEvent() method, 660

document.ids property, 661

document.images property, 664

document.images.length property, 665

Document.implementation property,
CD:1423

Document.importNode() method, CD:1424

document.lastModified property, 666

document.layers property, 667

document.layers.length property, 668

document.linkColor property, 669

document.links property, 670

document.links.length property, 671

document.onClick event handler, 672

document.onDblClick event handler, 673

document.onKeyDown event handler, 674

document.onKeyPress event handler, 675

document.onKeyUp event handler, 676

document.onLoad event handler, 677

document.onMouseDown event handler,
677

document.onMouseUp event handler, 678

document.onUnLoad event handler, 679

document.open() method, 680

document.plugins property, 681

document.plugins.length property, 682

document.referrer property, 682

document.releaseEvents() method, 683

document.routeEvent() method, 685

document.tags property, 687

document.title property, 690

document.unwatch() method, 690

document.URL property, 692

document.vlinkColor property, 692

document.watch() method, 693

document.write() method, 694

document.writeln() method, 695

DocumentCSS object, CD:1809

DocumentCSSList object, CD:1809

DocumentCSSList.getOverrideStyle()
method, CD:1809

documentElement property, CD:1412,
CD:1421

DocumentEvent object, CD:1731-1732

DocumentEvent.createEvent() method
object, CD:1732

DocumentFragment object, CD:1424-1425

documents

- anchors, code to iterate, CD:1526-1527
- applets, code to iterate, CD:1527-1528
- createDocument()** method, code,
CD:1432
- createDocumentFragment()** method,
CD:1416-1417, CD:1425
- createDocumentType()** method, code,
CD:1433
- createTextNode()** method, code,
CD:1420
- dynamic, 145-154
- elements, code to add, CD:1418
- events, code to pass, 660-661
- forms, code to iterate, CD:1530-1531,
CD:1540-1541
- hasFeature()** method, code, CD:1434
- layers, code to display, 668-669
- links, code to iterate, CD:1533
- opening, code, 680
- output streams, code to close, 651-652
- text, **find()** method
code, 1112
- title property, code to access, 690
- XML, comments, code to add, CD:1416

DocumentType object, CD:1425

DocumentType.entities property, CD:1426

DocumentType.internalSubset property,
CD:1426

DocumentType.name property, CD:1427

DocumentTypenotations property,
CD:1427-1428

DocumentType.publicId property, CD:1428
DocumentType.systemId property, CD:1429
dollar sign (\$) (pattern matching character), 87
DOM (Document Object Model), 12, 146-151
DOM Level 2 Core API, CD:1479
DOM Level 2 Core language bindings, CD:1403
DOM Level 2 Event Model, CD:1731
domain property, 636, 1109, CD:1525, CD:1530
DOMException object, CD:1429-1431
DOMException.code property, CD:1430
DOMImplementation object, CD:1431
DOMImplementation.createDocument() method, CD:1432
DOMImplementation.createDocumentType() method, CD:1433
DOMImplementation.hasFeature() method, CD:1433
DOMImplementationCSS object, CD:1810
DOMImplementationCSS.createCSSStyleSheet() method, CD:1810-1811
DOMSTRING SIZE ERR constant (DOMException object), CD:1429
dot (.) (pattern matching character), 86
dot notation, code to define event handlers, 93
double keyword, 333
double quotation marks (" "), 35, 1215
downloading
 Page Signer tool, 22
 Scripting Technologies (Microsoft), CD:1824
 Windows Script Host, CD:1927
 WSH (Windows Script Host) engine, 184
do:while loop, 283
do...while loop, 68-73, 332-333
DragDrop event, 91
DRAGDROP
 event (Event object), 698
 property, code, 726
Drive object, CD:1832-1842
drive parameter, CD:1879-1880
Drive property, CD:1845, CD:1852-1853, CD:1891, CD:1898
drive types, CD:1835-1836

DriveExists() method (FileSystemObject object), CD:1865, CD:1873
DriveLetter property, CD:1833-1835
drives
 A, 338-339, CD:1838
 C, CD:1839-1840
 checking, code, CD:1874
 D, CD:1873
 letters for files, code to obtain, CD:1852-1853
 moving to, code, 340-341
 number of, code to access, CD:1843
 retrieving, code, CD:1844-1845
 share names, code to return, CD:1841
 total size, code to display, CD:1841-1842
 volume name, code to return, CD:1842
Drives collection, CD:1842-1845, CD:1865, CD:1874
DriveType property, CD:1833-1836
dynamic documents, client-side scripting, 145-154
Dynamic HTML (DHTML), 4, 145-148, 151-154
dynamically building links, code, 1176

E

E property, 408, 416-417
e-mail, 26-27, 172-174, 1354-1361
Echo() method, CD:1929, CD:1934
ECMAScript (European Computer Manufacturers Association), 7, 10, 193
Edit menu commands, Preferences, 136
ELEMENT NODE constant (Node object), CD:1454
Element object, CD:1434-1435
Element.getAttribute() method, CD:1435
Element.getAttributeNode() method, CD:1436
Element.getAttributeNodeNS() method, CD:1436-1437
Element.getAttributeNS() method, CD:1437
Element.getElementsByTagName() method, CD:1438
Element.getElementsByTagNameNS() method, CD:1438
Element.hasAttribute() method, CD:1439
Element.hasAttributeNS() method, CD:1439

Element.removeAttribute() method,
CD:1440

Element.removeAttributeNode() method,
CD:1440-1441

Element.removeAttributeNS() method,
CD:1441

Element.setAttribute() method, CD:1442

Element.setAttributeNode() method,
CD:1442

Element.setAttributeNodeNS() method,
CD:1443

Element.setAttributeNS() method,
CD:1444

element.style.visibility property, 152

Element.tagName property, CD:1444

ElementCSSInlineStyle object, CD:1811

ElementCSSInlineStyle.style property,
CD:1812

elements

Document object, code to create,
CD:1435

documents, code to add, CD:1418

DOM (Document Object Model),
creating, 149

of forms, code to access, 764-765

getting by tag name, code,

CD:1422-1423, CD:1438-1439

hasAttribute() method, code, CD:1439

hasAttributeNS() method, code,
CD:1440

hasChildren() method, code, CD:1417

retrieving by identifier, code,

CD:1421-1422

elements array, 129-130

elements property, 761, CD:1546, CD:1549

elements.length property, 761, 765-766

ellipsis (...), (pattern matching character),
87

else keyword, code, 334

else statement, 334, 389, 490

else...if statement, 64

Embed object, 697

<embed> tag, 697

embedded objects, 654-655, 697

embeds array, 697

embeds property, 636, 654, 1109

enabledPlugin property (MimeType
object), 899-901

enableExternalCapture() method, 1085,
1111

enablePrivilege() method, 22

encodeURIComponent() function, 334-335, 365, 369

encodeURIComponentComponent() function, 335,
366, 370

encoding property, 761, CD:1770-1771

encryption methods, 1103-1104

enctype property, CD:1546, CD:1550

End event (WshRemote object),

CD:1972-1974

end of files, code to find, 1280-1281

End() method (Response object), 1335,
1340

ending script lines with semicolon (;), 31-32
engines

Scripting Engine (Microsoft), CD:1823

WSH (Windows Script Host), download-
ing, 184

Enterprise Server, server-side JavaScript,
170-176

entities property (DocumentType object),
CD:1425-1426

ENTITY NODE constant (Node object),
CD:1454

Entity object, CD:1445

ENTITY REFERENCE NODE constant
(Node object), CD:1454

entity references, code to create,
CD:1418-1419

Entity.notationName property, CD:1445

Entity.publicId property, CD:1446

Entity.systemId property, CD:1446

EntityReference object, CD:1447

enum keyword, 336

Enumerator object, 336-337

Enumerator.atEnd() method, 337

Enumerator.item() method, 338

Enumerator.moveFirst() method, 339

Enumerator.moveNext() method, 340

enumerators, code to move, 339-340

EnumNetworkDrives() method, CD:1964,
CD:1967

EnumPrinterConnection() method,
CD:1964, CD:1968

Environment property, CD:1990, CD:1993
environment variables

COMSPEC, CD:1955

deleting, code, CD:1959

displaying, code, CD:1956-1958

HOMEDRIVE, CD:1955

HOMEPATH, CD:1955

NUMBER OF PROCESSORS, CD:1955

OS, CD:1955

- PATH, CD:1955
- PATHEXT, CD:1955
- PROCESSOR ARCHITECTURE, CD:1955
- PROCESSOR IDENTIFIER, CD:1955
- PROCESSOR LEVEL, CD:1955
- PROCESSOR REVISION, CD:1955
- PROMPT, CD:1955
- retrieving, code, 1387
- ssjs_getCGIVariable() function, 1386-1387
- SYSTEMDRIVE, CD:1955
- SYSTEMROOT, CD:1955
- TEMP, CD:1955
- TMP, CD:1955
- WINDIR, CD:1955
- Windows, CD:1955
- environments**
 - ASP, 169
 - server-side, 158
- eof() method (File object), 1275, 1280-1281**
- equal comparison operator (==), 56-57**
- equal operator (==), 233-234**
- equals() method, 397400**
- error codes (e-mail failed deliveries), displaying, 1359-1360**
- Error event, 91, CD:1972-1974**
- ERROR event (Event object), 698**
- error messages, sending e-mail, 1360-1361**
- error() method, 1275, 1281-1282**
- Error object, 341-345**
- ERROR property, code to access, 727**
- Error property, CD:1972, CD:1975**
- Error.description property, 342**
- Error.message property, 343**
- Error.name property, 343-344**
- Error.number property, 344-345**
- errorCode() method (SendMail object), 1355, 1359-1360**
- errorMessage() method (SendMail object), 1355**
- errors**
 - break statements, 75
 - continue keyword, 73
 - continue statements, 75
 - ConversionError, 344
 - database connections, code to retrieve, 1205-1208, 1243-1245
 - Database/Function, 167
 - databases, 167
 - files, code to clear, 1278-1279
 - labels, 75
 - LiveWire Services, 167
 - RangeError, 344
 - ReferenceError, 344
 - RegExpError, 344
 - runtime, 344, 99-100
 - standard error output, CD:1940
 - Standard Server-Side JavaScript, 167
 - SyntaxError, 344
 - TypeError, 344
 - URIError, 344
 - Vendor, 167
- Errorsto property, 1355, 1361**
- escape sequences (strings), special characters, 35-36**
- escape() function, 345-346**
- escape() method, 366, 371**
- ESPN, 139**
- European Computer Manufacturers Association (ECMA), 7**
- eval() function, 152, 346-347, 372**
- eval() method, 366, 397, 400-401, 452-454**
- EvalError (runtime error), 99**
- event event, code, 1384**
- event functions, 1178-1179**
- event handlers, 89-90, 92**
 - Area object, 574
 - Area.onDbClick, 584-585
 - Area.onMouseOut, 585
 - Area.onMouseOver, 586
 - Button object, 598
 - Button.onBlur, 606
 - Button.onClick, 606-607
 - Button.onFocus, 607
 - Button.onMouseDown, 608
 - Button.onMouseUp, 608
 - Checkbox object, 615
 - Checkbox.onBlur, 627
 - Checkbox.onClick, 628-629
 - Checkbox.onFocus, 630
 - defining with dot notation, code, 93
 - Document object, 637
 - document.onClick, 672
 - document.onDbClick, 673
 - document.onKeyDown, 674
 - document.onKeyPress, 675
 - document.onKeyUp, 676
 - document.onLoad, 677
 - document.onMouseDown, 677
 - document.onMouseUp, 678
 - document.onUnload, 679

events, 92-95

FileUpload.onBlur, 753-754

FileUpload.onChange, 754

FileUpload.onFocus, 755

Frame.onBlur, 784

Frame.onFocus, 785

Frame.onMove, 786

Frame.onResize, 786

Image object, 808

Image.onAbort, 814

Image.onError, 815

Image.onKeyDown, 815

Image.onKeyPress, 816

Image.onKeyUp, 816

Image.onLoad, 816-817

invoking, 95

Layer.onBlur, 844

Layer.onFocus, 845

Layer.onLoad, 846

Layer.onMouseOut, 847

Layer.onMouseOver, 848

Link.onClick, 871

Link.onDbClick, 871-872

Link.onKeyDown, 872

Link.onKeyPress, 873

Link.onKeyUp, 874

Link.onMouseDown, 874

Link.onMouseOut, 875

Link.onMouseOver, 876

Link.onMouseUp, 877

onAbort, 92, 808, 814-815

onBlur, 92

 alert boxes, code to display, 606, 627-628

 Button object, 598

 Checkbox object, 615

 code to access, 784-785

 code, 844-845, 928-929, 1038-1039, 1072-1073, 1130

 Layer object, 844

 Password object, 923

 Radio object, 939

 Reset object, 953

 Select object, 974

 Submit object, 1030

 Text object, 1046

 Textarea object, 1063

onChange, 92

 code, 754-755, 985-986, 1056, 1073-1074

 Select object, 974

 Text object, 1046

 Textarea object, 1063

onClick, 92

 alert boxes, code to display, 607

 Button object, 598

 Checkbox object, 615, 630

 click events, code, 672

 code, 871, 947-948, 960

 Document object, 637

 messages, code to display, 629-630

 Radio object, 939

 Reset object, 953

 Submit object, 1030

 textbox contents, code to display, 1039-1040

 window.document property, 1108

onDbClick, 92

 Area object, 574, 585

 code, 872

 DbClick events, code, 673

 Document object, 637

 onDragDrop, 92

 window.document property, 1108
 code, 1130

onError, 92, 808, 815, 1131

onFocus, 92

 alert boxes, code to display, 607-608

 Button object, 598

 Checkbox object, 615

 cheese, code to select, 630-631

 code, 755-756, 845-846, 948-949, 1040-1041, 1131-1132

 Layer object, 845

 Password object, 923

 Radio object, 939

 Reset object, 953

 Select object, 975

 Submit object, 1030

 Text object, 1046

 Textarea object, 1063

onKeyDown, 92

 code, 815, 872, 1076

 Document object, 637

 Image object, 808

 KeyDown events, code, 674

 Textarea object, 1063

 window.document property, 1108

onKeyPress, 92

 code, 816, 873, 1076-1077

 Document object 637

- Image object, 808
- KeyPress events, code, 675
- Textarea object, 1063
- window.document property, 1108
- onKeyUp, 92
 - code, 816, 874, 1077-1078
 - Document object, 637
 - Image object, 808
 - KeyUp events, code, 676
 - Link object, 865
 - Textarea object, 1063
 - window.document property, 1108
- onLoad, 92
 - <body> tag, 118
 - code, 817, 846-847, 1132
 - Document object, 637
 - Image object, 808
 - Layer object, 846
 - Load events, code, 677
- onMouseDown, 92
 - Button object, 598
 - code, 874-875
 - Document object, 637
 - MouseDown events, code, 677-678
 - window.document property, 1108
- onMouseMove, 92
- onMouseOut, 92, 574
 - code, 585-586, 847-848, 875-876
 - Layer object, 847
- onMouseOver, 92
 - Area object, 574, 586-587
 - code, 848-849, 876-877
- onMouseUp, 92
 - Button object, 598
 - code, 877
 - Document object, 637
 - MouseUp events, code, 678-679
 - window.document property, 1108
- onMove, code, 786, 1132
- onMouseDown, alert boxes, code to display, 608
- onMouseUp, alert boxes, code to display, 609
- onReset, 92, 771-772
- onResize, 92, 786-787, 1133
- onSelect, 92
 - code, 1058, 1078-1079
 - Text object, 1046
 - Textarea object, 1063
- onSubmit, 92
- onUnload, 92, 637, 680, 1133
- Password object, 923
- Password.onBlur, 928
- Password.onFocus, 929
- Radio object, 939
- Radio.onBlur, 946
- Radio.onClick, 947
- Radio.onFocus, 948
- Reset object, 953
- Reset.onBlur, 959
- Reset.onClick, 960
- Reset.onFocus, 961
- return values, 94-95
- Select object, 974
- Select.onBlur, 984
- Select.onChange, 985
- Select.onFocus, 986
- Submit object, 1030
- Submit.onBlur, 1038
- Submit.onClick, 1039
- Submit.onFocus, 1040
- Text object, 1046
- Text.onBlur, 1054
- Text.onChange, 1055
- Text.onFocus, 1056
- Text.onSelect, 1057
- Textarea object, 1063
- Textarea.onBlur, 1072
- Textarea.onChange, 1073
- Textarea.onFocus, 1074
- Textarea.onKeyDown, 1075
- Textarea.onKeyPress, 1076
- Textarea.onKeyUp, 1077
- Textarea.onSelect, 1078
- timers, 95-96
- window.document property, 1108
- window.onBlur, 1129
- window.onDragDrop, 1130
- window.onError, 1130-1131
- window.onFocus, 1131
- window.onLoad, 1132
- window.onMove, 1132
- window.onResize, 1133
- window.onUnload, 1133
- Event keyword, code, 699-700**
- event listener, CD:1749-1755**
- event masks, 646-647**
- Event object, 698-699, 707, CD:1732-1733, CD:1748, CD:1756**
- event property (HTMLScriptElement object), CD:1655**
- Event.ABORT property, 720**
- Event.BLUR property, 721**

Event.bubbles property, CD:1734
Event.cancelable property, CD:1735
Event.CHANGE property, 722
Event.CLICK property, 723
Event.currentTarget property, CD:1736
event.data property, 701-702
Event.DBLCLICK property, 724
Event.DRAGDROP property, 725
Event.ERROR property, 726-727
Event.eventPhase property, CD:1736-1737
Event.FOCUS property, 728
event.height property, 702-703
Event.initEvent() method, CD:1737
Event.KEYDOWN property, 729
Event.KEYPRESS property, 730
Event.KEYUP property, 731
event.layerX property, 703-704
event.layerY property, 704
Event.LOAD property, 732
event.modifiers property, 705-706
Event.MOUSEDOWN property, 733
Event.MOUSEMOVE property, 734
Event.MOUSEOUT property, 735
Event.MOUSEOVER property, 736
Event.MOUSEUP property, 737
Event.MOVE property, 738
event.pageX property, 707
event.pageY property, 708
Event.preventDefault() method, CD:1738
Event.RESET property, 739
Event.RESIZE property, 741
event.screenX property, 709-710
event.screenY property, 711
Event.SELECT property, 742
Event.stopPropagation() method, CD:1739
Event.SUBMIT property, 743
event.target property, 712, CD:1739
Event.timeStamp property, CD:1740
event.type property, 713, CD:1734
Event.UNLOAD property, 744
event.unwatch() method, 715-716
event.watch() method, 716
event.which property, 718
event.width property, 718
EventException object, CD:1741-1743
EventException.code property, CD:1742
EventListener object, CD:1743
eventPhase property, CD:1733, CD:1737
events, 89

Abort, 90

ABORT (Event object), 698

addEventListener() method, code,
CD:1745-1746

Application object, 1178

Application OnEnd (Application object),
1178

Application OnStart (Application
object), 1178

assigning, code, CD:1799

Blur, 90

BLUR (Event object), 698

canceling with event handlers, 93-95

capture prevention, code, 684-685

capturing, 92-93, 647-648

Change, 90

CHANGE (Event object), 698

Click, 90

click, code, 600-605, 672

CLICK (Event object), 698

createEvent() method, code,
CD:1732-1734

DbiClick, 90, 673

DBLCLICK (Event object), 698

disableExternalCapture() method, code,
1106-1107

dispatchEvent() method (EventTarget
object), code, CD:1744-1746

DocumentEvent object, **createEvent()**
method, CD:1731-1732

DOM Level 2 Event Model, CD:1731

DragDrop, 91

DRAGDROP (Event object), 698

enableExternalCapture() method, code,
1111

End (WshRemote object),
CD:1972-1974

ERROR (Event object), 698

Error, 91, CD:1972-1974

event, code, 1384

Focus, 91

FOCUS (Event object), 698

handled (Event object), 698

initEvent() method, code,
CD:1737-1738

KeyDown, 91, 674

KEYDOWN (Event object), 698

KeyPress, 91, 675

KEYPRESS (Event object), 699

KeyUp, 91, 676

KEYUP (Event object), 699

Load, 91, 677

LOAD (Event object), 699

- mouse, code, CD:1748-1749, CD:1752
 - MouseDown, 91, code, 677-678
 - MOUSEDOWN (Event object), 699
 - MouseMove, 91
 - MOUSEMOVE (Event object), 699
 - MouseOut, 91
 - MOUSEOUT (Event object), 699
 - MouseOver, 91
 - MOUSEOVER (Event object), 699
 - MouseUp, 91, 678-679
 - MOUSEUP (Event object), 699
 - Move, 91
 - MOVE (Event object), 699
 - mutation, code, CD:1757-1759
 - myButton, 93
 - ObjectContext object, 1308
 - ObjectContext.OnTransactionAbort, 1309
 - ObjectContext.OnTransactionCommit, 1310
 - onclick property, 93
 - OnTransactionAbort, code, 1308-1310
 - OnTransactionCommit, 1308-1310
 - passing, 577-578, 625, 660-661, 685-686
 - releasing, 683-684
 - removeEventListener() method, code, CD:1747
 - Reset, 91
 - RESET (Event object), 699
 - Resize, 91
 - RESIZE (Event object), 699
 - routeEvent() method, 686-687
 - Select, 91
 - SELECT (Event object), 699
 - sending, code, CD:1799
 - Session object, 1379
 - Session OnEnd (Session object), 1379
 - Session OnStart (Session object), 1379
 - Session.Session_OnEnd, 1384
 - Session.Session_OnStart, 1384
 - Session_OnEnd, code, 1384
 - Start, CD:1972, CD:1976-1977
 - Submit, 91, 94
 - SUBMIT (Event object), 699
 - timed, code to create, 95-96
 - timers, 95-96
 - types, CD:1995
 - Unload, 91, 680
 - UNLOAD (Event object), 699
 - user interface, code, CD:1761, CD:1763
 - Web browsers, 90-91
 - window.captureEvents() method, code, 1095-1096
 - window.releaseEvents() method, code, 1149
 - window.routeEvent() method, code, 1153-1155
 - WshRemote object, CD:1972
 - WshRemote.End, CD:1973
 - WshRemote.Error, CD:1974
 - WshRemote.Start, CD:1976
 - EventTarget object, CD:1744**
 - EventTarget.addEventListener() method, CD:1745**
 - EventTarget.dispatchEvent() method, CD:1746**
 - EventTarget.removeEventListener() method, CD:1747**
 - Excel, multiplication tables, code to create, 242**
 - exceptions, 97-101, 344**
 - exec() method, 467, 477-478**
 - Exec() method, CD:1990, CD:1994**
 - exec(str) method, 89**
 - Execute() method, 1368-1370, CD:1972, CD:1976**
 - execute() method, 1199, 1205 1234, 1242**
 - Exists() method, CD:1825-1827, CD:1960**
 - exists() method (File object), 1275, 1282**
 - exp() method, 408, 417-418**
 - ExpandEnvironmentStrings() method, CD:1990, CD:1994-1995**
 - expiration() method, 1193, 1196**
 - expires attribute, cookies, 137**
 - Expires property, code, 1341**
 - ExpiresAbsolute property, code, 1341**
 - exponential notation, 34**
 - export keyword, 347-348**
 - expressions, evaluating, 103**
 - extends keyword, 348**
 - ExtensionName() method (FileSystemObject object), CD:1865**
 - extensions of files, .wsf, 180**
 - external events, 1106-1107, 1111**
 - External Libraries application setting (Server Side JavaScript Application Manager), 165**
- ## F
- \f escape sequence, 36**
 - f filelist command-line parameter, 161**

face property, CD:1505-1506,

CD:1544-1545

false keyword, 349

favorite foods, code to display, 243-244

fgColor property, 636, 656, 1109

fields, 1319-1320

File object, 174-176, 1274-1275,

CD:1845-1863

File property, 1183, 1186

File System Objects (FSO), CD:1833

file systems, code to display, CD:1836-1837

File.toString() method, 1277

File.clearError() method, 1278

File.close() method, 1279

File.constructor property, 1279-1280

File.eof() method, 1280

File.error() method, 1281

File.exists() method, 1282

File.flush() method, 1283

File.getLength() method, 1284

File.getPosition() method, 1284-1285

File.open() method, 1285

File.prototype property, 1286-1287

File.read() method, 1288

File.readByte() method, 1288-1289

File.readLine() method, 1289

File.setPosition() method, 1290

File.stringToByte() method, 1291

File.unwatch() method, 1292

File.watch() method, 1294

File.write() method, 1295

File.writeByte() method, 1296

File.writeln() method, 1297

FileExists() method (FileSystemObject object), CD:1865, CD:1874-1875

filename parameter, CD:1870-1871,

CD:1875, CD:1883, CD:1890

filename property (Plugin object), 934-936

filenames, retrieving and displaying,

CD:1854

files

binary data, code to write, 1296-1297

carriage returns, code to write,
1297-1298

checking, code, CD:1875

checking for existence, code, 1282

closing, code, 1279

content-type, code to set, 1177

content.type, code to delete, 1274

cookies.txt, 136

copying, code, CD:1848, CD:1867

creation dates, code to retrieve,

CD:1848-1849

cscript.exe, options, 185

dates last accessed, code, CD:1849-1850

dates last modified, code, CD:1850-1851

deleting, code, CD:1851-1852, CD:1871

drive letters, code to obtain,

CD:1852-1853

end of, code to find, 1280-1281

Enterprise Server, 174-176

Errors, code to clear, 1278-1279

existence of, code to check, 1282

extensions, .wsf, 180

JAR (Java Archive), 22

jscript.dll, CD:1824

moving to new directory, code, CD:1853

opening, code, 1286

scrrun.dll, CD:1824

short names, code to access, CD:1858

short paths, code to obtain,

CD:1858-1859

sizes, code to obtain, CD:1859-1860

strings, code, 1295-1298, CD:1855-1856

types, code to obtain, CD:1860

web.html, 141

wscript.exe, options, 185

WSH (Windows Script Host), code, 180

XUL structure, 155

Files property, CD:1891, CD:1899

FileSystem property, CD:1833,

CD:1836-1837

FileSystemObject object, CD:1833-1834,

CD:1864-1890

**fileType parameter (blobImage() method),
1190**

FileUpload object, 745-747

FileUpload.blur() method, 747

FileUpload.focus() method, 748

FileUpload.form property, 749-750

**FileUpload.handleEvent() method,
750-751**

FileUpload.name property, 752-753

FileUpload.onBlur event handler, 753-754

FileUpload.onChange event handler, 754

FileUpload.onFocus event handler, 755

FileUpload.select() method, 756

FileUpload.type property, 757

FileUpload.unwatch() method, 758

FileUpload.value property, 759

FileUpload.watch() method, 760

final keyword, 349

finally block, 100-101

find() method, 1085, 1112

finding

- C drive available space, code, CD:1834
- D drive, code, CD:1873
- files (end of), code, 1280-1281
- HTML elements, code, 640-641
- 's' fruits in dictionary, code, CD:1827
- tmp folder, code, CD:1876-1877

firstChild property, CD:1455, CD:1459

fixed() method (String object), 498

Flags field (ClientCertificate collection), 1319

float keyword, 349

float values, CD:1782-1786

floating-point numbers, 34

floor() method, 408, 418-419

flush() function, 1298

flush() method, 1275, 1283

Flush() method, 1335, 1342

flushing buffers to pages, code, 1298

focus

- check boxes, code to apply, 623
- select boxes, code to remove, 977
- setting to upload box, code, 748
- Submit button, code, 1032-1034
- text areas, code to remove, 1065-1066
- text boxes, code to remove, 1048
- windows, code, 1092-1093, 1112-1113

Focus event, 91

FOCUS

- event (Event object), 698
- property, code to access, 728-729

focus() method, 1086

- Button object, 598
- buttons, code to shift focus, 601-602
- check boxes, focus, code to apply, 623
- Checkbox object, 615
- code, 782, 943-944, 1112-1113, CD:1484, CD:1593, CD:1719
- FileUpload object, 746
- Frame object, 779
- HTMLAnchorElement object, CD:1481
- HTMLInputElement object, CD:1586
- HTMLSelectElement object, CD:1659
- HTMLTextAreaElement object, CD:1715
- Password object, 923
- Radio object, 939
- Reset object, 952
- select boxes, code to place cursor, 978-979

Select object, 975

Submit button, code to set focus, 1034

Submit object, 1030

text areas, code to place cursor, 1067-1068

text boxes, code to place cursor, 1050-1051

Text object, 1046

Textarea object, 1063

Folder object, CD:1891-1892, CD:1902

folder objects, code to retrieve, CD:1884

Folder.Attributes property, CD:1892-1893

Folder.Copy() method, CD:1893-1894

Folder.DateCreated property, CD:1894

Folder.DateLastAccessed property, CD:1895

Folder.DateLastModified property, CD:1896

Folder.Delete() method, CD:1897

Folder.Drive property, CD:1898

Folder.Files property, CD:1899

Folder.IsRootFolder property, CD:1899

Folder.Move() method, CD:1900

Folder.Name property, CD:1901

Folder.ParentFolder property, CD:1902

Folder.Path property, CD:1903

Folder.ShortName property, CD:1903-1904

Folder.ShortPath property, CD:1904-1905

Folder.Size property, CD:1905-1906

Folder.SubFolders property, CD:1906-1907

Folder.Type property, CD:1907

Folder.Exists() method (FileSystemObject object), CD:1865, CD:1876-1877

foldername parameter, CD:1872, CD:1876, CD:1885

folders

- deleting, code, CD:1872
- parent, code to get names, CD:1885
- root, code to return, CD:1839-1840
- special types, CD:2012-2013
- tmp, code to find, CD:1876-1877

Folders collection, CD:1908-1909

Folders.Add() method, CD:1909

Folders.Count property, CD:1910

Folders.Item property, CD:1911

fontcolor() method (String object), 498

fontFamily property, 649, 662, 688, 995, 1008-1009

fonts, 652, 1009

fontSize() method (String object), 498

fontSize property, 649, 662, 688, 995, 1010

fontStyle property, 649, 662, 688, 995, 1010-1011
fontWeight property, 649, 662, 688, 995, 1011-1012
food orders, simulating, code, 266
foods, code to display, 243-244
for keyword, 350
for loop, 66-67, 73, 283, 350
ForAppending constant (iomode parameter), CD:1855
force parameter, CD:1871-1872
Form array, 125-128
form array, Forms object, code to access, 762-763
Form collection, 1318, 1322
Form objects, 658-659, 761-764
form processing, output, 127
form property

- Button object, 597
- button 's parent, code to access, 602-603
- checkbox box parent, code to access, 624
- Checkbox object, 615
- code, 794, 926, CD:1515, CD:1601, CD:1719
- FileUpload object, 745
- forms, code to access, 979-980, 1035-1036, 1051-1052, 1069
- Hidden object, 793
- HTMLButtonElement object, CD:1513
- HTMLFieldSetElement object, CD:1543
- HTMLInputElement object, CD:1586
- HTMLIsIndexElement object, CD:1600
- HTMLLabelElement object, CD:1602
- HTMLLegendElement object, CD:1605
- HTMLObjectElement object, CD:1624
- HTMLOptionElement object, CD:1641
- HTMLSelectElement object, CD:1659
- HTMLTextAreaElement object, CD:1715
- Password object, 922
- Radio object, 938
- Reset object, 952
- Select object, 975
- Submit object, 1030
- Text object, 1046
- Textarea object, 1064

<form> tag, 761-763, 124-125
Form.action property, 763
Form.elements property, 764
Form.elements.length property, 765
Form.encoding property, 766-767

Form.handleEvent() method, 767
Form.length property, 768-769
Form.method property, 769
Form.name property, 770
Form.onReset property, 771
Form.OnSubmit property, 772
Form.reset() method, 773
Form.submit() method, 774
Form.target property, 775
Form.unwatch() method, 776
Form.watch() method, 777
format parameter, CD:1855, CD:1890
formats

- SELECT queries, code, 1213
- var x = new Array(n);, 40

formatting

- <a> tag, code, 1192
- tag, code, 1189-1191
- SELECT query, code, 1248-1249

formKey property, 1327-1329
formName property (window.document property), 1109
forms

- accessing, code, 126-127, 979-980, 1035-1036, 1051-1052, 1069
- actions, code to access, 656-657
- client-side validations, code, 131-135
- elements, code to access, 129, 764-765
- HTML code, 176
- indexes, 127
- information, code to write, 130-131
- iterating, code, CD:1530-1531, CD:1540-1541
- names, code to access, 657-658
- processing, 124-136, 177
- property, 636, 1109, CD:1525
- property array, form names, code to access, 657-658
- submitting, 176-177, 1033

Forms array, forms, code to access, 126-127
ForReading constant (iomode parameter), CD:1855
forward() method, 799-802, 1086, 1114, 1117, 1171
forward slash (/), 86, 587, CD:2016
ForWriting constant (iomode parameter), CD:1855
for...in loop, 70-73, 283
for...in object, 351
Frame object, 778-779

frame property, CD:1688, CD:1697
 <frame> tag, 778
 Frame.blur() method, 779-780
 Frame.clearInterval() method, 780
 Frame.clearTimeout() method, 781
 Frame.document property, 781
 Frame.focus() method, 782
 Frame.frames property, 782-783
 Frame.length property, 783
 Frame.name property, 784
 Frame.onBlur event handler, 784
 Frame.onFocus event handler, 785
 Frame.onMove event handler, 786
 Frame.onResize event handler, 786
 Frame.parent property, 787
 Frame.print() method, 787
 Frame.self property, 788
 Frame.setInterval() method, 788-789
 Frame.setTimeout() method, 789
 Frame.top property, 790
 Frame.unwatch() method, 790-791
 Frame.watch() method, 791
 Frame.window property, 792
 frameBorder property, CD:1554-1556,
 CD:1570-1572
 frames, 1144, 1170-1171
 frames property, 778, 783, 1087, 1114
 free space, code to retrieve, CD:1837-1838
 FreeSpace property, CD:1833,
 CD:1837-1838
 From property, 1355, 1362
 fromCharCode() method (String object),
 498
 fruit dictionaries, CD:1825-1827
 FSO (File System Objects), CD:1833
 FullName property, CD:1929, CD:1934,
 CD:2005-2007, CD:2020-2021
 function argument (Array object), 263-264
 Function keyword, 80, 352
 Function object, 353
 Function() constructor, 352
 Function.apply() method, 353-354
 Function.arguments property, 354-355
 Function.arity property, 355-356
 Function.call() method, 356
 Function.callee property, 357
 Function.caller property, 358
 Function.constructor property, 359
 Function.length property, 360
 Function.prototype property, 360-361
 Function.toSource() method, 361-362

Function.toString() method, 362
 Function.valueOf() method, 363
 functions
 { } (curly brackets), 77
 addClient(), 1176, 1316
 addResponseHeader(), 1176-1177
 Application.Application_OnEnd event,
 1178
 Application.Application_OnStart event,
 1179
 Arguments, code to access with argu-
 ments array, 79
 as objects, 80
 blob(), 1188
 call by reference, 77
 call by value, 77-78
 call by value versus call by reference, 78
 callC(), 1192
 callC() (registered), code, 1192-1193
 changeState(), 152
 checkNum, 136
 checkPhone, 136
 dailyTask(), 95
 debug(), 1273
 decodeURI(), 329, 367
 decodeURIComponent(), 330-331, 368
 deleteResponseHeader(), 1274
 displayArguments(), 79
 encodeURI(), 334, 369
 encodeURIComponent(), 335, 370
 escape(), 345-346
 eval(), 152, 346-347, 372
 EventException object, code,
 CD:1741-1743
 EventListener object, code, CD:1743
 flush(), 1298
 GetObject, 364-365
 getOptionValue(), 1299-1300
 getOptionValueCount(), 1300
 Global.decodeURI(), 366
 Global.decodeURIComponent(), 367
 Global.encodeURI(), 368
 Global.encodeURIComponent(), 369
 Global.eval(), 371-372
 Global.isNaN(), 375
 isNaN(), 375-376, 388-389
 monitorSpecs(), 39
 redirect(), 1176, 1316
 registerCFunction, 1317
 ResetFields(), 76
 return statements, 79

returning values, 79-80
ScriptEngine(), 494
ScriptEngineBuildVersion(), 494-495
ScriptEngineMajorVersion(), 495
ScriptEngineMinorVersion(), 495-496
setColor(), 104
setType(), 104
showCar, 130
showColor, 130
ssjs_generateClientID(), 1385-1386
ssjs_getCGIVariable(), 1386-1387
ssjs_getClientID(), 1387
String(), 497
syntax, 77
ValidatePersonalInfo, 135
variables, declaring, 38
write(), 1398-1399

G

g (pattern attribute), 88
games, pop-up windows, 139
GATEWAY INTERFACE variable
(ServerVariables collection), 1325
general information resources, 28-29
GET method, 124-125
GetAbsolutePathName() method
(FileSystemObject object), CD:1865,
CD:1877
getAttribute() method, CD:1434-1436,
CD:1443
getAttributeNode() method, CD:1434-1436
getAttributeNodeNS() method, CD:1434,
CD:1437
getAttributeNS() method, CD:1434,
CD:1437
GetBaseName() method
(FileSystemObject object), CD:1865,
CD:1878
getComputedStyle() method (ViewCSS
object), CD:1818-1819
getCounterValue() method
(CSSPrimitiveValue object),
CD:1781-1782
getDate() method (Date object), 286,
289-290
getDay() method (Date object), 286,
290-291
GetDrive() method (FileSystemObject
object), CD:1833-1834, CD:1865,
CD:1879
GetDriveName() method
(FileSystemObject object), CD:1865,
CD:1879-1880
getElementById() method, 149, 152,
CD:1413
getElementsByName() method, CD:1525,
CD:1531-1532
getElementsByTagName() method,
CD:1413, CD:1434, CD:1438
getElementsByTagNameNS() method,
CD:1413, CD:1434, CD:1438-1439
GetExtensionName() method
(FileSystemObject object), CD:1880-1881
GetFile() method (FileSystemObject
object), CD:1865, CD:1882
GetFileName() method (FileSystemObject
object), CD:1865, CD:1882-1883
getFloatValue() method, CD:1781-1783
GetFolder() method (FileSystemObject
object), CD:1865, CD:1883-1884
getFullYear() method (Date object), 286,
291-292, 305
getHours() method (Date object), 286, 292
getItem() method (VBAArray object), 551
GetLastError() method, 1368-1370
getLength() method, 1275, 1284
getMember() method, 397, 402
getMilliseconds() method (Date object),
286, 292-293
getMinutes() method (Date object), 287,
293
getMonth() method (Date object), 287,
294-295
getNamedItem() method, CD:1448-1449
getNamedItemNS() method, CD:1449
GetObject function, 364-365
GetObject() method, CD:1929, CD:1935
getOptionValue() function, 1299-1300
getOptionValueCount() function, 1300
getOverrideStyle() method, CD:1809-1810
GetParentFolderName() method
(FileSystemObject object), CD:1865,
CD:1885
getPosition() method, 1275, 1285
getPropertyCSSValue() method,
CD:1792-1794
getPropertyPriority() method, CD:1792,
CD:1795
getPropertyValue() method,
CD:1793-1796
getRectValue() method, CD:1781-1783

getRGBColor() method, code,
 CD:1781-1782
 getRGBColorValue() method, code,
 CD:1781, CD:1784
 getSeconds() method (Date object), 287,
 295-296
 getSelection() method, 637, 1108
 getSlot() method, 397, 403
 GetSpecialFolder() method
 (FileSystemObject object), CD:1865,
 CD:1885-1886
 getStringValue() method, CD:1781,
 CD:1784
 GetTempName() method
 (FileSystemObject object), CD:1865,
 CD:1886-1887
 getTime() method (Date object), 287,
 296-297
 getTimezoneOffset() method (Date object),
 287, 297
 getUTCDate() method (Date object), 287,
 298
 getUTCDay() method (Date object), 287,
 298-299
 getUTCFullYear() method (Date object),
 287, 300
 getUTCHours() method (Date object), 287,
 300-301
 getUTCMilliseconds() method (Date
 object), 287, 301
 getUTCMinutes() method (Date object),
 287, 301-302
 getUTCMonth() method (Date object),
 287, 302-304
 getUTCSeconds() method (Date object),
 287, 304
 getWindow() method, 397, 403
 getWrappedException() method, 395-396
 getYear() method (Date object), 287,
 305-306
 Global object, 365-366
 global property, 467, 478
 global variables, 38-39
 Global.decodeURI() function, 366
 Global.decodeURIComponent() function,
 367
 Global.encodeURI() function, 368
 Global.encodeURIComponent() function,
 369
 Global.escape() method, 370
 Global.eval() function, 371-372

Global.Infinity keyword, 373
 Global.isFinite() method, 374
 Global.isNaN() function, 375
 Global.Nan object, 376
 Global.Number() method, 376-377
 Global.parseFloat method, 377
 Global.parseInt method, 378
 Global.String method, 379
 Global.undefined property, 380
 Global.unescape method, 380
 GN subfield (ClientCertificate collection),
 1320
 go method (window.history property), 1117
 go() method, 799, 802
 goto keyword, 381
 greater than operator (>), 57, 235-236
 greater than or equal operator (>=),
 236-237
 greater than or equal to comparison opera-
 tor (>=), 58
 green property (RGBColor object),
 CD:1816-1817
 grocery inventory, code to display, 251-252

H

-h command-line parameter, 161
 handled events (Event object), 698
 handleEvent
 method (Window object), 1086
 property (Link object), 865
 handleEvent() method
 Area object, 573
 Button object, 598
 Checkbox object, 615
 clicks on pages, code, 1052-1053,
 1070-1071
 code, 603-605, 751-752, 767-768, 811
 events, 660-661, 625
 FileUpload object, 746
 Form object, 761
 Image object, 808
 Layer object, 822
 Password object, 923
 Radio object, 939
 Reset object, 952
 Select object, 975
 Submit object, 1030
 Text object, 1046
 Textarea object, 1063
 window.document property, 1108
 handlers. *See* event handlers

handling, items from dictionaries, code, CD:1828
hasAttribute() method, CD:1434, CD:1439
hasAttributeNS() method, CD:1434, CD:1440
hasAttributes() method, CD:1455, CD:1459-1460
hasChildNodes() method (Node object), CD:1455
hasChildren() method, code, CD:1417
hasFeature() method, 149, CD:1431, CD:1434
hash property, 573, 579-580, 865-868, 885-887
hash symbol (#), 578, 885
<head> tag, 126
headers property, CD:1674, CD:1680
height parameter, 1135, 1190
height property, 811-812, 971-972, CD:1494, CD:1572
 Event object, 698
 HTMLAppletElement object, CD:1490
 HTMLIFrameElement object, CD:1570
 HTMLImageElement object, CD:1577
 HTMLObjectElement object, CD:1624
 HTMLTableCellElement object, CD:1674
 Image object, 807
Hello World, 185-187
help. *See* resources; support
hexadecimal, 43, 380
Hidden object, 793-794
Hidden.form property, 794
Hidden.name property, 795
Hidden.type property, 795
Hidden.unwatch() method, 796
Hidden.value property, 797
Hidden.watch() method, 798
hierarchies, 10-16, 181
HIERARCHY REQUEST ERR constant (DOMException object), CD:1429
histories, forward() method, code, 1114
history
 JavaScript, development of, 6-7
 programming languages, development of, 5-6
History object, 799
history object, data tainted by default, 21
history property, 117-118, 1087
History.back() method, 800
History.current property, 800

History.forward() method, 801
History.go() method, 802
History.length property, 802-803
History.next property, 803
History.previous property, 804
History.unwatch() method, 805
History.watch() method, 806
home, 1086, 1118
HOMEDRIVE (Windows environment variable), CD:1955
HOMEPATH (Windows environment variable), CD:1955
host property
 Area object, displaying, code, 573, 581
 code, 868-869, 888, 1374
 Link object, 865
 Location object, 885
 server object, 1373
hostname property
 Area object, 573, 582-583
 code, 869-870, 889-890, 1375
 Link object, 865
 Location object, 885
 server object, 1373
HotJava, 108, 115
Hotkey property, CD:2005, CD:2008
hotkeys parameter (Window object), 1135
hours argument, 286, 326
href attribute (tag), 1190
href property, 870-871, 890-891, CD:1484-1485, CD:1500
 Area object, 573, 583-584
 CSSImportRule object, CD:1772
 HTMLAnchorElement object, CD:1480
 HTMLAreaElement object, CD:1497
 HTMLBaseElement object, CD:1503
 HTMLLinkElement object, CD:1609
 Link object, 865
 Location object, 886
hrefLang property, CD:1480, CD:1485, CD:1609, CD:1612
hspace property
 code, 812-813, CD:1494, CD:1580-1581, CD:1632
 HTMLAppletElement object, CD:1490
 HTMLImageElement object, CD:1577
 HTMLObjectElement object, CD:1624
 Image object, 807
HTML (Hypertext Markup Language)
 attributes, language or type, 32
 code, applets, loading and changing text, 122-123

- DHTML (Dynamic HTML), 145
- elements, finding, code, 640-641
- forms, code, 176
- tags, 111-112, 118, 124-125, 146-148, 151, CD:1928
 - <!-- -->, 33
 - <!--, 33
 - >, 33
 - <a>>, code to format, 1192
 - <body>, 781
 - <embed>, 697
 - <form>, 761, 763
 - <frame>, 778
 - , 807-808, 1189-1191
 - <input>, 745, 793, 922, 938
 - <job>, 180
 - <layer>, 824
 - <script>, 22, 32-33, 159, 180
 - <select>, 1299
 - <server>, 158-159
 - <table>, 1199
 - <tags:head>, 126
 - <tagshtml:script>, 155
- <html> tag, 148
- HTMLAnchorElement object, CD:1480-1489
- HTMLAnchorElement.accessKey property, CD:1481
- HTMLAnchorElement.blur() method, CD:1482
- HTMLAnchorElement.charset property, CD:1482
- HTMLAnchorElement.coords property, CD:1483
- HTMLAnchorElement.focus() method, CD:1484
- HTMLAnchorElement.href property, CD:1484
- HTMLAnchorElement.hrefLang property, CD:1485
- HTMLAnchorElement.name property, CD:1485
- HTMLAnchorElement.rel property, CD:1486
- HTMLAnchorElement.rev property, CD:1486
- HTMLAnchorElement.shape property, CD:1487
- HTMLAnchorElement.tabIndex property, CD:1488
- HTMLAnchorElement.target property, CD:1488
- HTMLAnchorElement.type property, CD:1489
- HTMLAppletElement object, CD:1489-1497
- HTMLAppletElement.align property, CD:1490
- HTMLAppletElement.alt property, CD:1491
- HTMLAppletElement.archive property, CD:1492
- HTMLAppletElement.code property, CD:1492
- HTMLAppletElement.codeBase property, CD:1493
- HTMLAppletElement.height property, CD:1493
- HTMLAppletElement.hspace property, CD:1494
- HTMLAppletElement.name property, CD:1495
- HTMLAppletElement.object property, CD:1495
- HTMLAppletElement.vspace property, CD:1496
- HTMLAppletElement.width property, CD:1496
- HTMLAreaElement object, CD:1497-1503
- HTMLAreaElement.accessKey property, CD:1498
- HTMLAreaElement.alt property, CD:1498
- HTMLAreaElement.coords property, CD:1499
- HTMLAreaElement.href property, CD:1500
- HTMLAreaElement.noHref property, CD:1500
- HTMLAreaElement.shape property, CD:1501
- HTMLAreaElement.tagIndex property, CD:1501-1502
- HTMLAreaElement.target property, CD:1502
- HTMLBaseElement object, CD:1503-1505
- HTMLBaseElement.href property, CD:1503-1504
- HTMLBaseElement.target property, CD:1504
- HTMLBaseFontElement object, CD:1505-1507

- HTMLBaseFontElement.color property, CD:1505-1506
- HTMLBaseFontElement.face property, CD:1506
- HTMLBaseFontElement.size property, CD:1507
- HTMLBodyElement object, CD:1507-1512
- HTMLBodyElement.aLink property, CD:1508
- HTMLBodyElement.background property, CD:1509
- HTMLBodyElement.bgColor property, CD:1509
- HTMLBodyElement.link property, CD:1510
- HTMLBodyElement.text property, CD:1511
- HTMLBodyElement.vLink property, CD:1511
- HTMLBRElement object, CD:1512-1513
- HTMLBRElement.clear property, CD:1512-1513
- HTMLButtonElement object, CD:1513-1518
- HTMLButtonElement.accessKey property, CD:1514
- HTMLButtonElement.disabled property, CD:1514-1515
- HTMLButtonElement.form property, CD:1515
- HTMLButtonElement.name property, CD:1516
- HTMLButtonElement.tabIndex property, CD:1516
- HTMLButtonElement.type property, CD:1517
- HTMLButtonElement.value property, CD:1517
- HTMLCollection object, CD:1518-1521, CD:1532
- HTMLCollection.item() method, CD:1519
- HTMLCollection.length property, CD:1520
- HTMLCollection.namedItem() method, CD:1520-1521
- HTMLDirectoryElement object, CD:1521-1522
- HTMLDirectoryElement.compact property, CD:1522
- HTMLDivElement object, CD:1522-1523
- HTMLDivElement.align property, CD:1523
- HTMLDListElement object, CD:1524-1525
- HTMLDListElement.compact property, CD:1524
- HTMLDocument object, CD:1525-1526
- HTMLDocument.anchors property, CD:1526
- HTMLDocument.applets property, CD:1527
- HTMLDocument.body property, CD:1528
- HTMLDocument.close() method, CD:1528
- HTMLDocument.cookie property, CD:1529
- HTMLDocument.domain property, CD:1530
- HTMLDocument.forms property, CD:1530
- HTMLDocument.getElementsByName() method, CD:1531
- HTMLDocument.images property, CD:1532
- HTMLDocument.links property, CD:1533
- HTMLDocument.open() method, CD:1533
- HTMLDocument.referrer property, CD:1534
- HTMLDocument.title property, CD:1535
- HTMLDocument.URL property, CD:1535
- HTMLDocument.write() method, CD:1536
- HTMLDocument.writeln() method, CD:1537
- HTMLDOMImplementation object, CD:1537-1538
- HTMLDOMImplementation.createHTMLDocument() method, CD:1538
- HTMLElement object, CD:1539-1540
- HTMLElement.className property, CD:1540
- HTMLElement.dir property, CD:1540
- HTMLElement.id property, CD:1541
- HTMLElement.lang property, CD:1541-1542
- HTMLElement.title property, CD:1542
- HTMLEncode() method, code, 1368-1370
- HTMLFieldSetElement object, CD:1543-1544
- HTMLFieldSetElement.form property, CD:1543
- HTMLFontElement object, CD:1544-1546
- HTMLFontElement.color property, CD:1545

- HTMLFontElement.face property, CD:1545
- HTMLFontElement.size property, CD:1546
- htmlFor property, CD:1602-1604, CD:1655
- HTMLFormElement object, CD:1546-1554
- HTMLFormElement.acceptCharset property, CD:1547
- HTMLFormElement.action property, CD:1548
- HTMLFormElement.elements property, CD:1549
- HTMLFormElement.enctype property, CD:1549
- HTMLFormElement.length property, CD:1550
- HTMLFormElement.method property, CD:1551
- HTMLFormElement.name property, CD:1551-1552
- HTMLFormElement.reset() method, CD:1552
- HTMLFormElement.submit() method, CD:1553
- HTMLFormElement.target property, CD:1553
- HTMLFrameElement object, CD:1554-1560
- HTMLFrameElement.contentDocument property, CD:1555
- HTMLFrameElement.frameBorder property, CD:1555
- HTMLFrameElement.longDesc property, CD:1556
- HTMLFrameElement.marginHeight property, CD:1557
- HTMLFrameElement.marginWidth property, CD:1557
- HTMLFrameElement.name property, CD:1558
- HTMLFrameElement.noResize property, CD:1558-1559
- HTMLFrameElement.scrolling property, CD:1559
- HTMLFrameElement.src property, CD:1560
- HTMLFrameSetElement object, CD:1560-1562
- HTMLFrameSetElement.cols property, CD:1561-1562
- HTMLHeadElement object, CD:1562-1563
- HTMLHeadElement.profile property, CD:1563
- HTMLHeadingElement object, CD:1564-1565
- HTMLHeadingElement.align property, CD:1564
- HTMLHRElement object, CD:1565-1568
- HTMLHRElement.align property, CD:1566
- HTMLHRElement.noShade property, CD:1566
- HTMLHRElement.size property, CD:1567
- HTMLHRElement.width property, CD:1567
- HTMLHtmlElement object, CD:1568-1569
- HTMLHtmlElement.version property, CD:1569
- HTMLIFrameElement object, CD:1569-1577
- HTMLIFrameElement.align property, CD:1570
- HTMLIFrameElement.contentDocument property, CD:1571
- HTMLIFrameElement.frameBorder property, CD:1571
- HTMLIFrameElement.height property, CD:1572
- HTMLIFrameElement.longDesc property, CD:1573
- HTMLIFrameElement.marginHeight property, CD:1573
- HTMLIFrameElement.marginWidth property, CD:1574
- HTMLIFrameElement.name property, CD:1574
- HTMLIFrameElement.scrolling property, CD:1575
- HTMLIFrameElement.src property, CD:1576
- HTMLIFrameElement.width property, CD:1576
- HTMLImageElement object, CD:1577-1585
- HTMLImageElement.align property, CD:1578
- HTMLImageElement.alt property, CD:1578-1579
- HTMLImageElement.border property, CD:1579
- HTMLImageElement.height property, CD:1580

HTMLImageElement.hspace property,
CD:1580
HTMLImageElement.isMap property,
CD:1581
HTMLImageElement.longDesc property,
CD:1581
HTMLImageElement.lowSrc property,
CD:1582
HTMLImageElement.src property,
CD:1583
HTMLImageElement.useMap property,
CD:1583
HTMLImageElement.vspace property,
CD:1584
HTMLImageElement.width property,
CD:1584
HTMLInputElement object, CD:1585-1600
HTMLInputElement.accept property,
CD:1587
HTMLInputElement.accessKey property,
CD:1587
HTMLInputElement.align property,
CD:1588
HTMLInputElement.alt property,
CD:1588
HTMLInputElement.blur() method,
CD:1589
HTMLInputElement.checked property,
CD:1589-1590
HTMLInputElement.click() method,
CD:1590
HTMLInputElement.defaultChecked
property, CD:1591
HTMLInputElement.defaultValue prop-
erty, CD:1591-1592
HTMLInputElement.disabled property,
CD:1592
HTMLInputElement.focus() method,
CD:1593
HTMLInputElement.form property,
CD:1593
HTMLInputElement.maxLength property,
CD:1594
HTMLInputElement.name property,
CD:1594
HTMLInputElement.readOnly property,
CD:1595
HTMLInputElement.select() method,
CD:1596
HTMLInputElement.size property,
CD:1596

HTMLInputElement.src property,
CD:1597
HTMLInputElement.tabIndex property,
CD:1597-1598
HTMLInputElement.type property,
CD:1598
HTMLInputElement.useMap property,
CD:1599
HTMLInputElement.value property,
CD:1599
HTMLIsIndexElement object,
CD:1600-1602
HTMLIsIndexElement.form property,
CD:1601
HTMLIsIndexElement.prompt property,
CD:1601
HTMLLabelElement object,
CD:1602-1604
HTMLLabelElement.accessKey property,
CD:1603
HTMLLabelElement.form property,
CD:1603
HTMLLabelElement.htmlFor property,
CD:1604
HTMLLegendElement object,
CD:1604-1607
HTMLLegendElement.accessKey prop-
erty, CD:1605
HTMLLegendElement.align property,
CD:1606
HTMLLegendElement.form property,
CD:1606
HTMLLIElement object, CD:1607-1609
HTMLLIElement.type property, CD:1608
HTMLLIElement.value property, CD:1608
HTMLLinkElement object, CD:1609-1615
HTMLLinkElement.charset property,
CD:1610
HTMLLinkElement.disabled property,
CD:1610
HTMLLinkElement.href property,
CD:1611
HTMLLinkElement.hreflang property,
CD:1612
HTMLLinkElement.media property,
CD:1612
HTMLLinkElement.rel property, CD:1613
HTMLLinkElement.rev property,
CD:1613-1614
HTMLLinkElement.target property,
CD:1614

HTMLLinkElement.type property, CD:1615	HTMLObjectElement.standby property, CD:1633
HTMLMapElement object, CD:1615-1617	HTMLObjectElement.tabIndex property, CD:1633
HTMLMapElement.areas property, CD:1616	HTMLObjectElement.type property, CD:1634
HTMLMapElement.name property, CD:1616-1617	HTMLObjectElement.useMap property, CD:1634
HTMLMenuElement object, CD:1617-1618	HTMLObjectElement.vspace property, CD:1635
HTMLMenuElement.compact property, CD:1618	HTMLObjectElement.width property, CD:1636
HTMLMetaElement object, CD:1618-1621	HTMLLOListElement object, CD:1636-1638
HTMLMetaElement.content property, CD:1619	HTMLLOListElement.compact property, CD:1637
HTMLMetaElement.httpEquiv property, CD:1620	HTMLLOListElement.start property, CD:1637-1638
HTMLMetaElement.name property, CD:1620	HTMLLOListElement.type property, CD:1638
HTMLMetaElement.scheme property, CD:1621	HTMLOptGroupElement object, CD:1639-1641
HTMLModElement object, CD:1622-1623	HTMLOptGroupElement.disabled proper- ty, CD:1639
HTMLModElement.cite property, CD:1622	HTMLOptGroupElement.label property, CD:1640
HTMLModElement.dateTime property, CD:1623	HTMLOptionElement object, CD:1641-1647
HTMLObjectElement object, CD:1624-1636	HTMLOptionElement.defaultSelected property, CD:1642
HTMLObjectElement.align property, CD:1625, CD:1629	HTMLOptionElement.disabled property, CD:1642-1643
HTMLObjectElement.archive property, CD:1625	HTMLOptionElement.form property, CD:1643
HTMLObjectElement.border property, CD:1626	HTMLOptionElement.index property, CD:1644
HTMLObjectElement.code property, CD:1627	HTMLOptionElement.label property, CD:1645
HTMLObjectElement.codeBase property, CD:1627	HTMLOptionElement.selected property, CD:1645
HTMLObjectElement.codeType property, CD:1628	HTMLOptionElement.text property, CD:1646
HTMLObjectElement.contentDocument property, CD:1628-1629	HTMLOptionElement.value property, CD:1647
HTMLObjectElement.declare property, CD:1630	HTMLParagraphElement object, CD:1647-1649
HTMLObjectElement.form property, CD:1630	HTMLParagraphElement.align property, CD:1648
HTMLObjectElement.height property, CD:1631	HTMLParamElement object, CD:1649-1652
HTMLObjectElement.hspace property, CD:1631-1632	HTMLParamElement.name property, CD:1649-1650
HTMLObjectElement.name property, CD:1632	

HTMLParamElement.type property,
CD:1650
HTMLParamElement.value property,
CD:1651
HTMLParamElement.valueType property,
CD:1651
HTMLPreElement object, CD:1652-1653
HTMLPreElement.width property,
CD:1653
HTMLQuoteElement object,
CD:1653-1654
HTMLQuoteElement.cite property,
CD:1654
HTMLScriptElement object,
CD:1654-1659
HTMLScriptElement.charset property,
CD:1655
HTMLScriptElement.defer property,
CD:1656
HTMLScriptElement.event property,
CD:1657
HTMLScriptElement.htmlFor property,
CD:1657
HTMLScriptElement.src property,
CD:1657
HTMLScriptElement.text property,
CD:1658
HTMLScriptElement.type property,
CD:1658
HTMLSelectElement object,
CD:1659-1670
HTMLSelectElement.add() method,
CD:1660
HTMLSelectElement.blur() method,
CD:1661
HTMLSelectElement.disabled property,
CD:1661
HTMLSelectElement.focus() method,
CD:1662
HTMLSelectElement.form property,
CD:1663
HTMLSelectElement.length property,
CD:1663
HTMLSelectElement.multiple property,
CD:1664
HTMLSelectElement.name property,
CD:1665
HTMLSelectElement.options property,
CD:1665
HTMLSelectElement.remove() method,
CD:1666

HTMLSelectElement.selectedIndex prop-
erty, CD:1667
HTMLSelectElement.size property,
CD:1667-1668
HTMLSelectElement.tabIndex property,
CD:1668
HTMLSelectElement.type property,
CD:1669
HTMLSelectElement.value property,
CD:1669-1670
HTMLStyleElement object, CD:1670-1673
HTMLStyleElement.disabled property,
CD:1671
HTMLStyleElement.media property,
CD:1671-1672
HTMLStyleElement.type property,
CD:1672
HTMLTableCaptionElement object,
CD:1673-1674
HTMLTableCaptionElement.align prop-
erty, CD:1673
HTMLTableCellElement object,
CD:1674-1683
HTMLTableCellElement.abbr property,
CD:1675-1676
HTMLTableCellElement.align property,
CD:1676
HTMLTableCellElement.axis property,
CD:1676
HTMLTableCellElement.bgColor prop-
erty, CD:1677
HTMLTableCellElement.cellIndex prop-
erty, CD:1677
HTMLTableCellElement.ch property,
CD:1678
HTMLTableCellElement.chOff property,
CD:1678
HTMLTableCellElement.colSpan property,
CD:1679
HTMLTableCellElement.headers property,
CD:1680
HTMLTableCellElement.height property,
CD:1680
HTMLTableCellElement.noWrap prop-
erty, CD:1681
HTMLTableCellElement.rowSpan prop-
erty, CD:1681
HTMLTableCellElement.scope property,
CD:1682
HTMLTableCellElement.vAlign property,
CD:1682

- HTMLTableCellElement.width property, CD:1683
- HTMLTableColElement object, CD:1683-1688
- HTMLTableColElement.align property, CD:1684
- HTMLTableColElement.ch property, CD:1685
- HTMLTableColElement.chOff property, CD:1686
- HTMLTableColElement.span property, CD:1686
- HTMLTableColElement.vAlign property, CD:1687
- HTMLTableColElement.width property, CD:1687-1688
- HTMLTableElement object, CD:1688-1702
- HTMLTableElement.align property, CD:1689
- HTMLTableElement.bgColor property, CD:1690
- HTMLTableElement.border property, CD:1690
- HTMLTableElement.caption property, CD:1691
- HTMLTableElement.cellPadding property, CD:1692
- HTMLTableElement.cellSpacing property, CD:1692
- HTMLTableElement.createCaption() method, CD:1693
- HTMLTableElement.createTFoot() method, CD:1693-1694
- HTMLTableElement.createTHead() method, CD:1694
- HTMLTableElement.deleteCaption() method, CD:1695
- HTMLTableElement.deleteRow() method, CD:1695
- HTMLTableElement.deleteTFoot() method, CD:1696
- HTMLTableElement.deleteTHead() method, CD:1696
- HTMLTableElement.frame property, CD:1697
- HTMLTableElement.insertRow() method, CD:1698
- HTMLTableElement.rows property, CD:1698
- HTMLTableElement.rules property, CD:1699
- HTMLTableElement.summary property, CD:1699
- HTMLTableElement.tBodies property, CD:1700
- HTMLTableElement.tFoot property, CD:1701
- HTMLTableElement.tHead property, CD:1701
- HTMLTableElement.width property, CD:1702
- HTMLTableRowElement object, CD:1702-1709
- HTMLTableRowElement.align property, CD:1703
- HTMLTableRowElement.bgColor property, CD:1704
- HTMLTableRowElement.cells property, CD:1705
- HTMLTableRowElement.ch property, CD:1705
- HTMLTableRowElement.chOff property, CD:1706
- HTMLTableRowElement.deleteCell() method, CD:1706
- HTMLTableRowElement.insertCell() method, CD:1707
- HTMLTableRowElement.rowIndex property, CD:1708
- HTMLTableRowElement.sectionRowIndex property, CD:1708
- HTMLTableRowElement.vAlign property, CD:1709
- HTMLTableSectionElement object, CD:1709-1714
- HTMLTableSectionElement.align property, CD:1710
- HTMLTableSectionElement.ch property, CD:1711
- HTMLTableSectionElement.chOff property, CD:1711
- HTMLTableSectionElement.deleteRow() method, CD:1712
- HTMLTableSectionElement.deleteSection() method, CD:1712
- HTMLTableSectionElement.insertRow() method, CD:1713
- HTMLTableSectionElement.insertSection() method, CD:1713
- HTMLTableSectionElement.rows property, CD:1713

HTMLTableSectionElement.vAlign property, CD:1714
HTMLTextAreaElement object, CD:1714-1724
HTMLTextAreaElement.accessKey property, CD:1715-1716
HTMLTextAreaElement.blur() method, CD:1716
HTMLTextAreaElement.cols property, CD:1717
HTMLTextAreaElement.defaultValue property, CD:1717
HTMLTextAreaElement.disabled property, CD:1718
HTMLTextAreaElement.focus() method, CD:1718-1719
HTMLTextAreaElement.form property, CD:1719
HTMLTextAreaElement.name property, CD:1720
HTMLTextAreaElement.readOnly property, CD:1720
HTMLTextAreaElement.rows property, CD:1721
HTMLTextAreaElement.select() method, CD:1721-1722
HTMLTextAreaElement.tabIndex property, CD:1722
HTMLTextAreaElement.type property, CD:1723
HTMLTextAreaElement.value property, CD:1723
HTMLTitleElement object, CD:1724-1725
HTMLTitleElement.text property, CD:1725
HTMLULListElement object, CD:1725-1727
HTMLULListElement.compact property, CD:1726
HTMLULListElement.type property, CD:1727
<html:script> tag, 155
HTTP ACCEPT LANGUAGE variable (ServerVariables collection), 1325
HTTP ACCEPT variable (ServerVariables collection), 1325
HTTP COOKIE variable (ServerVariables collection), 1325
<HTTP HeaderName> variable (ServerVariables collection), 1325
HTTP REFERER variable (ServerVariables collection), 1325

HTTP USER AGENT variable (ServerVariables collection), 1325
httpEquiv property, CD:1618-620
HTTPS KEYSIZE variable (ServerVariables collection), 1325
HTTPS KEYSIZE variable (ssjs_getCGIVariable() function), 1386
HTTPS SECRETKEYSIZE variable (ServerVariables collection), 1325
HTTPS SECRETKEYSIZE variable (ssjs_getCGIVariable() function), 1386
HTTPS SERVER ISSUER variable (ServerVariables collection), 1325
HTTPS SERVER SUBJECT variable (ServerVariables collection), 1325
HTTPS variable (ServerVariables collection), 1325
HTTPS variable (ssjs_getCGIVariable() function), 1386
hyperlinks, 636, 1109, CD:1525
 building dynamically, code, 1176
 colors, code, 692-693
 creating, code, 642-643, 683
 displaying, code, 671
 iterating, code, CD:1533
 Lamborghini, 95
Hypertext Markup Language. *See* HTML

I

-i inputfile command-line parameter, 161
i (pattern attribute), 88
I subfield (ClientCertificate collection), 1320
IconLocation property, CD:2005, CD:2009
icons, pop-up message boxes, CD:1996
id property (HTMLElement object), CD:1539
identifier property (Counter object), CD:1767-1768
identifiers, elements, code to retrieve, CD:1421-1422
identity operator (===), 58, 234-235
IDs, style sheets, code to create, 663-664
ids property, 636, 662-663
IE (Internet Explorer), 563
if statement, 59, 62-63, 218-219, 381-382
if...else statement, 63-64, 382-383
ignoreCase property, 467, 479
IIS (Internet Information Server), 24, 157, 169, 176-178, 1175

- <ilayer> tag, 147
- Image object, 807-809
- image object, 19-21
- Image.border property, 809
- Image.complete property, 810
- Image.handleEvent() method, 811
- Image.height property, 811
- Image.hspace property, 812
- Image.lowsrc property, 813
- Image.name property, 814
- Image.onAbort event handler, 814
- Image.onError event handler, 815
- Image.onKeyDown event handler, 815
- Image.onKeyPress event handler, 816
- Image.onKeyUp event handler, 816
- Image.onLoad event handler, 816-817
- Image.src property, 817
- Image.unwatch() method, 818
- Image.vspace property, 819
- Image.watch() method, 819
- Image.width property, 820
- images, 664-665
- images property, 636, 1109, CD:1525
- images property array, 664-665
- imageX property, 1327, 1330
- imageY property, 1327, 1330
- tag, 148, 807-808, 1189-1191
- implementation property, CD:1413, CD:1423-1424
- implements keyword, 383
- import keyword, 383-384
- importing nodes, code, CD:1424
- importNode() method (Document object), CD:1413
- in keyword, 384-385
- increment operator (++), 226-227
- index property (Array object), 249, 253-254, CD:1641, CD:1644
- INDEX SIZE ERR constant (DOMException object), CD:1429
- indexes, 40-41, 127, CD:1450
- indexOf() method, 177, 498
- Infinity keyword, 385-386
- Infinity property, 365, 373-374
- information, handling, 28-30, 128-131, 142
- initEvent() method, CD:1733, CD:1737-1738
- Initial Page application setting (Server Side JavaScript Application Manager), 164
- initMouseEvent() method (Event object), CD:1748, CD:1756
- initUIEvent() method (UIEvent object), CD:1761
- innerHeight property, 1087, 1119, 1135
- innerWidth property, 1087, 1120-1121, 1135
- input property, 250, 254, 467, 480
- <input> tag, 745, 793, 922, 938
- insertBefore() method, CD:1455, CD:1461
- insertCell() method, CD:1703, CD:1707
- insertData() method, CD:1407, CD:1410
- inserting rows, code, 1226-1227
- insertRow() method
 - code, 1226-1227, CD:1698, CD:1713
 - Cursor object, 1220
 - HTMLTableElement object, CD:1689
 - HTMLTableSectionElement object, CD:1710
- insertRule(), CD:1775-1777, CD:1803
- insertRule(index) method (CSSStyleSheet object), CD:1801
- installations, plug-ins array, 119
- INSTANCE ID variable (ServerVariables collection), 1325
- INSTANCE META PATH variable (ServerVariables collection), 1325
- instance properties, 1030-1031, 1064-1065
- instanceof keyword, 386-387
- instances
 - DbPool object, code to create, 1262-1263
 - of vehicle objects, 8-9
 - properties, code to display, 975-976, 1047
 - Sproc object, code to create, 1389
 - Window object, code to create, 1088-1090
- instantiating Comment object, CD:1412
- int argument (Array object), 249
- int keyword, 387
- integers, 34
- Intel processor, code to alert when on, 199-200
- interactive e-mail, 26-27
- Interactive property, CD:1929, CD:1936
- interface keyword, 387
- interfaces
 - CGIs (common gateway interfaces), 5
 - CSSStyleSheet, 149
 - user interface events, code, CD:1761-1763
 - XUL (XML-based User Interface Language), 154-156

internalSubset property, CD:1425-1427

Internet Explorer (IE), 108, 111, 113-114, 563

Internet Information Server (IIS), 24, 157, 1175

intervals, 1097, 1163

INUSE ATTRIBUTE ERR constant (DOMException object), CD:1430

INVALID ACCESS ERR constant (DOMException object), CD:1430

INVALID CHARACTER ERR constant (DOMException object), CD:1429

INVALID MODIFICATION ERR constant (DOMException object), CD:1430

INVALID STATE ERR constant (DOMException object), CD:1430

inventory (grocery), 41-43, 251-252

invoking

event handlers, 95

stored procedures, code, 1250

iomode parameter, CD:1855, CD:1890

ip property, 1327, 1331

iPlanet Enterprise, Web site, 157

IsClientConnected property, code, 1342

isFinite() method, 366, 374-375, 387-388

ismap parameter (blobImage() method), 1190

isMap property, CD:1577, CD:1581

isNaN()

function, 375-376, 388-389

method (Global object), 366

IsReady property, CD:1833, CD:1838

IsRootFolder property, CD:1891, CD:1900

Issuer field (ClientCertificate collection), 1320

isSupported() method, CD:1456, CD:1461-1462

isValid() method, 1301-1302

italic text, 652

italics() method (String object), 498

item() method

code, CD:1450

CSSRuleList object, CD:1790

CSSStyleDeclaration object, CD:1792

CSSValueList object, CD:1807

document.all property array, 639

drives, accessing, code, 338-339

Enumerator object, 337

HTML elements, code to find, 640

HTMLCollection object, CD:1518

invoking, code, CD:1796,

CD:1807-1808

iterating, code, CD:1791

Item() method

Dictionary object, CD:1825,

CD:1828-1829

Item() property, CD:1827-1828,

CD:1844-1845

system directory path, code to display, CD:1957

WScript parameters, displaying, code, CD:1946-1947

WshArguments object, CD:1944

WshEnvironment object, CD:1954

Item property

arguments, code to display,

CD:1962-1963

code, CD:1911-1912, CD:2014-2015, CD:2018

Dictionary object, CD:1825

Drives collection, CD:1843

Files object, CD:1861-1864

Folders collection, CD:1909

WshNamed object, CD:1960

WshSpecialFolders object, CD:2013

WshUnnamed object, CD:2016

J

JAR files (Java Archive), 22

Java

applets, call() method

code, 116-117, 122-123, 398-399

JavaScript, comparing, 13

methods, code to access, 121

Java Capabilities API, Netscape DevEdge Web site, 22

java object, 389-390

java property (Packages object), 460

Java Security dialog box, 22

Java Virtual Machines (JVMs), 108

javaArray object, 390-391

javaArray.length property, 391

javaArray.toString() method, 392

javaClass object, 392-393

javaEnabled() method, 904, 907-908

javaObject object, 393-394

javaPackage object, 394-395

JavaScript

advantages, 23-25

applets, code to load, 401

- call() method, code, 399
- client-side object hierarchy, 11
- core language, 193
- development history, 6-7
- ECMAScript (ECMA-262), 7
- enabling through Server Side JavaScript
 - Application Manager, 162
- Java, comparing, 13
- JScript, comparing, 6-7, CD:1824
- server-side object hierarchy, 13
- Web browser support, 110-111
- when to use, 25

JavaScript Application Compiler (jsac), 160

<job> tag, 180, CD:1928

join() method, 42, 250, 254-255

jsac (JavaScript Application Compiler), 160-161

JScript, 169

- Application object, 168
- arrays, converting to Visual Basic arrays, CD:1829
- ASPErrors object, 168
- code, form processing, 177
- core language, 193
- detecting, code, 194
- ECMAScript (ECMA-262), 7
- JavaScript, comparing, 6-7, CD:1824
- Microsoft application support, CD:1823
- ObjectContext object, 168
- Request object, 168
- Response object, 168
- runtime, 15
- RunTime object hierarchy, 15
- scripting engine version information, code, 113-114
- Server object, 168
- server-side, 168
- Session object, 168
- Scripting Engine, 112
- versions, code to display, 196
- Web browser support, 112-113
- WSH (Windows Script Host), 7

jscript.dll file, CD:1824

JSErrors object, 395

JSException.getWrappedException() method, 396

JSObject object, 396-397

JSObject.call() method, 398

JSObject.equals() method, 399-400

JSObject.eval() method, 400

JSObject.getMember() method, 401-402

JSObject.getSlot() method, 403

JSObject.getWindow() method, 403

JSObject.removeMember() method, 404

JSObject.setMember() method, 405

JSObject.setSlot() method, 405

JSObject.toString() method, 406

jumps, continue statement, 73

JVMs (Java Virtual Machines), 108

K

Key property, CD:1825, CD:1829-1830

KeyDown event, 91, 674

KEYDOWN

- event (Event object), 698

- property, code to access, 729-730

KeyPress event, 91, 675

KEYPRESS

- event, 699

- object, 730-731

keys, CD:1830-1831, CD:1997-1999

Keys() method (Dictionary object), CD:1825, CD:1830-1831

keystrokes, CD:2001-2002

KeyUp event, 91, 676

KEYUP

- event, 699

- property, code to access, 731-732

keywords

- abstract, 241

- boolean, 272-273

- break, 71-73, 280-281

- byte, 281-282

- case, 282

- char, 282

- class, 282

- const, 283

- continue, 73-74, 283

- debugger, 328

- default, 331

- do, 332

- double, 333

- else, 334

- enum, 336

- Event, 699-700

- export, 347-348

- extends, 348

- false, 349

- final, 349

- float, 349

- for, 350
- Function, 80, 352
- function, 352
- Global.Infinity, 373
- goto, 381
- implements, 383
- import, 383-384
- in, 384-385
- Infinity, 385-386
- instanceof, 386-387
- int, 387
- interface, 387
- label, 406-407
- long, 407-408
- native, 436
- new, 86
- null, 36, 438
- package, 459
- private, 465-466
- protected, 466
- public, 466
- return, 493
- short, 496
- static, 496-497
- super, 537
- synchronized, 539
- this, 539-541
- throw, 541-543
- throws, 543
- transient, 543
- true, 544
- var, 37-38, 549-550
- volatile, 559-560

L

- l charset command-line parameter, 161
- L subfield (ClientCertificate collection), 1320
- label argument, 280-283
- label keyword, 406-407
- label property, CD:1639-1641, CD:1645
- labeling statements, 72
- labels, 71-75, 280-281
- labels and continue statement, code, 74
- Lamborghini link, 95
- lang property, CD:1539, CD:1542
- language attribute, 33, 57
- language bindings, DOM Level 2 Core, CD:1403

- language independent, defined, 27
- language property, 904, 908
- languages. *See also* HTML; JScript
 - core language, 193
 - DHTML (Dynamic HTML), 4, 145-148, 151-154
 - ECMAScript (European Computer Manufacturers Association), 7, 10, 193
 - XUL (XML-based User Interface Language), 154-156
- lastChild property, CD:1455, CD:1462
- lastIndex property, 250, 255-256, 467, 481
- lastIndexOf() method (String object), 498
- lastMatch property, 467, 482
- lastModified property, 636, 666-667, 1109
- lastName object, 86
- lastParen property, 467, 482-483
- Layer object, 667-668, 821-822
- layer object, origin verification, 19
- <layer> tag, 146, 824
- Layer.above property, 823
- Layer.background property, 824
- Layer.below property, 825
- Layer.bgColor property, 826
- Layer.captureEvents() method, 827
- Layer.clip.bottom property, 828
- Layer.clip.height property, 829
- Layer.clip.left property, 830
- Layer.clip.right property, 831
- Layer.clip.top property, 832
- Layer.clip.width property, 833
- Layer.document property, 834
- Layer.handleEvent() method, 835
- Layer.left property, 837
- Layer.load() method, 838
- Layer.moveAbove() method, 838-839
- Layer.moveBelow() method, 839
- Layer.moveBy() method, 840
- Layer.moveTo() method, 841
- Layer.moveToAbsolute() method, 842
- Layer.name property, 843
- Layer.onBlur event handler, 844
- Layer.onFocus event handler, 845
- Layer.onLoad event handler, 846
- Layer.onMouseOut event handler, 847
- Layer.onMouseOver event handler, 848
- Layer.pageX property, 849
- Layer.pageY property, 850
- Layer.parentLayer property, 851
- Layer.releaseEvents() method, 852
- Layer.resizeBy() method, 853-854

Layer.resizeTo() method, 854
Layer.routeEvent() method, 855
Layer.siblingAbove property, 856-857
Layer.siblingBelow property, 857
Layer.src property, 858
Layer.top property, 859
Layer.unwatch() method, 860
Layer.visibility property, 861
Layer.watch() method, 862
Layer.zIndex property, 864
layers, 146-147, 153-154, 668-669
layers property, 636, 1109
layers property array, 667-668
layerX property (Event object), 698
layerY property, 698, 704-705
lbound() method (VBAArray object), 551
LCID property, 1379, 1383
left property, 822, 837, CD:1812-1814
leftContext property, 467, 483-484
length, arrays, 42
length attribute (Array object), 42
length method (toSring() object), 390
Length property, CD:1944, CD:1954, CD:1960
length property
 accessing, code, 989
 applets, 116-117, 645
 Arguments object, 243, 247-249
 arguments, code to display, CD:1963
 Array object, 250, 256-257
 CharacterData object, CD:1407
 code, 360, 391-392, 783, 803, 936,
 CSSRuleList object, CD:1790
 CSSStyleDeclaration object, CD:1792
 CSSValueList object, CD:1807
 embedded objects, code to list, 655
 environment variables, code to display,
 CD:1958
 Form object, 658-659, 761
 Frame object, 778
 Function object, 353
 getting, code, CD:1796-1797
 History object, 799
 HTMLCollection object, CD:1518
 HTMLFormElement object, CD:1546
 HTMLSelectElement object, CD:1659
 images, number of, code to display, 665
 iterating, code, CD:1520
 javaArray object, 390
 JSObject object, 397
 layers in documents, code to display,
 668-669

 links, code to display, 671
 Plugin object, 934
 plug-ins, accessing number of, 119
 reading, code, CD:1791-1792, CD:1808
 select box names, code to retrieve,
 982-983
 Select object, 975
 String object, 500
 text data length, code to check,
 CD:1408-1409
 Window, 1087
 window.history property, 1117
 WshSpecialFolders object, CD:2013
 WshUnnamed object, CD:2016
less than operator (<), 57-58, 228-229
less than or equal operator (<=), 58,
231-232
letter parameter (DriveExists() method),
CD:1873
letters, variables, 37
libraries, 1317, CD:1824
limitations, windows, 142, 182
Line property
 ASPErrors object, 1183
 code, 1186, CD:1916-1917,
 CD:1981-1982
 TextStream object, CD:1913
 WshRemoteError object, CD:1979
lineHeight property, 649, 662, 688, 995,
1012-1013
lines (script), ending with semicolons, 32
lineStyleType property (Style object), 995
link colors, code to set, 669-670
Link object, 21, 864-866
link property, 670-671, CD:1507, CD:1510
link() method (String object), 498
Link.handleEvent() method, 866
Link.hash property, 867
Link.host property, 868
Link.hostname property, 869
Link.href property, 870
Link.onClick event handler, 871
Link.onDbClick event handler, 871-872
Link.onKeyDown event handler, 872
Link.onKeyPress event handler, 873
Link.onKeyUp event handler, 874
Link.onMouseDown event handler, 874
Link.onMouseOut event handler, 875
Link.onMouseOver event handler, 876
Link.onMouseUp event handler, 877
Link.pathname property, 877-878

Link.port property, 878-879
Link.protocol property, 879
Link.search property, 880
Link.target property, 881
Link.text property, 882
Link.unwatch() method, 883
Link.watch() method, 884
linkColor property, 636, 669-670, 1109
links. *See* hyperlinks
listings. *See also* code
 Drives collection, accessing, CD:1843
 embedded objects, code, 654-655
 Enumerator object, moving to the beginning of the collection, 654
 File object, creating, CD:1846
 Object() object, 452
listStyle property, CD:1767-1769
listStyleType property, 649, 662, 688, 1013-1014
literal characters, 87
LiveConnect, 120, 122-123
LiveWire Services error, 167
LN2 property, 408, 420
LN10 property, 408, 419
Load event, 91, 677
LOAD
 event (Event object), 699
 property, code, 733
load() method, 822, 838
LOCAL ADDR variable (ServerVariables collection), 1325
local variables, 38
localName property, CD:1455, CD:1462-1463
location argument (ActiveXObject object), 242
location object, 19-21
Location object, 885-886
location parameter (Window object), 1135
location property, 1087, 1122
Location.hash property, 886
Location.host property, 887-888
Location.hostname property, 889
Location.href property, 890
Location.pathname property, 891
Location.port property, 892
Location.protocol property, 893
Location.reload() method, 894
Location.replace() method, 895
Location.search property, 895
Location.unwatch() method, 896-897
Location.watch() method, 898

locationbar property, 1087, 1122
locationbar.visible property, 1123
lock() method, 1301-1303, 1312-1313, 1373-1375
Lock() object, 1300-1301
Lock.constructor property, 1301
Lock.isValid() method, 1302
Lock.lock() method, 1303
Lock.prototype property, 1303-1304
Lock.unlock() method, 1305
Lock.unwatch() method, 1305
Lock.watch() method, 1307
locking applications, code, 1303-1305, 1313
locks, code to verify, 1302
log() method, 408, 421
LOG2E property, 408, 423
LOG10E property, 408, 422
LogEvent() method, CD:1990, CD:1995-1996
logical AND operator (&&), 54-55, 211-212
logical NOT operator (!), 54-55, 205-206
logical OR operator (||), 54-55, 222-223
LOGON USER variable (ServerVariables collection), 1325
long keyword, 407-408
longDesc property, CD: 1554-1556, CD:1570, CD:1573, CD:1577, CD:1582
loops
 break keyword, 71-73
 break statements, 75
 continue keyword, 73-74
 continue statement and labels, code, 74
 continue statements, 73-75
 do...while, 68-70, 73, 283, 332-333
 for, 66-67, 73, 283, 350
 for...in, 70-73, 283
 labels, 71-75
 nested, continue statements, 75
 stopping, code, 337-338
 structure in continue statement, 283
 while, 67-68, 73, 283
 with statement, 75-76
lowsrc property, 807, 813, CD:1577-1578, CD:1582

M

Macintosh, data tainting, enabling for Netscape Navigator, 20
majorErrorCode() method
 code, 1205-1206
 Connection object, 1199

- database connection errors, code to retrieve, 1243, 1264
- database object, 1234
- DbPool object, 1257
- majorErrorMessage() method**
 - code, 1206-1207
 - Connection object, 1199
 - database connection errors, code to retrieve, 1243-1244, 1265
 - database object, 1234
 - DbPool object, 1257
- managers, Server Side JavaScript**
 - Application Manager, 161-167**
- MapNetworkDrive() method, CD:1964, CD:1968-1969**
- MapPath() method, 1368, 1371**
- mappings, network drives, code to retrieve, CD:1967**
- marginBottom property, 649, 662, 688, 996, 1014-1015**
- marginHeight property, CD:1554, CD:1557, CD:1570, CD:1573-1574**
- marginLeft property, 649, 662, 688, 996, 1015**
- marginRight property, 649, 662, 688, 996, 1016**
- margins() method, 650, 663, 689, 995, 1017**
- marginTop property, 649, 662, 688, 996, 1018**
- marginWidth property, CD:1554, CD:1557-1558, CD:1570, CD:1574**
- masks, events, 646-647**
- match() method (String object), 499**
- match(regExpObj) method, 88**
- matches, positions, code to display, 253-256**
- matching patterns, 86-89**
- Math object, 34-35, 46, 408-409**
- Math.abs() method, 46, 409**
- Math.acos() method, 46, 410**
- Math.asin() method, 46, 411**
- Math.atan() method, 46, 412**
- Math.atan2() method, 46, 413**
- Math.ceil() method, 46, 414**
- Math.cos() method, 46, 415**
- Math.E constant, 34**
- Math.E property, 416**
- Math.exp() method, 46, 417**
- Math.floor() method, 46, 418**
- Math.LN2, 34, 420**
- Math.LN10 constant, 34, 419**
- Math.log() method, 46, 420-421**
- Math.LOG2E, 34, 422**
- Math.LOG10E, 35, 421-422**
- Math.max() method, 46, 423**
- Math.min() method, 46, 424**
- Math.PI constant, 35**
- Math.pow() method, 46, 426**
- Math.random() method, 46, 427**
- Math.round() method, 46, 428**
- Math.sin() method, 46, 429**
- Math.sqrt() method, 46, 430**
- Math.SQRT1_2**
 - constant, 35
 - property, 431
- Math.SQRT2**
 - constant, 35
 - property, 432
- Math.tan() method, 46, 432-433**
- Math.toSource() method, 433**
- Math.toString() method, 434**
- Math.unwatch() method, 435**
- Math.watch() method, 435**
- max() method, 409, 423-424**
- MAX VALUE property, 439-442**
- MaxConn parameter, 171**
- maxConn parameter, 1238, 1256, 1259, 1262**
- maxLength property, CD:1586, CD:1594**
 - code, CD:1612-1613, CD:1672
 - CSSImportRule object, CD:1772
 - CSSMediaRule object, CD:1775
 - HTMLLinkElement object, CD:1609
 - HTMLStyleElement object, CD:1670
 - media types, code to set, CD:1777
 - reading, code, CD:1774
- media types, code to set, CD:1775**
- menubar property, 1087, 1124, 1135**
- menubar.visible property, code, 1124-1125**
- message boxes, pop-up, CD:1996**
- message property (Error object), 342**
- messages, 343, 629-630**
- metaKey property, CD:1748, CD:1753**
- method attribute, GET or POST, 124-125**
- method property**
 - code, 769-770, CD:1551
 - evaluating, code, 1331-1332
 - Form object, 761
 - HTMLFormElement object, CD:1547
 - Request object, 1327
- methods**
 - Abandon(), 1379-1380
 - abs() (Math object), 408

- `acos()`, code, 408-411
- `ActiveXObject()`, Dictionary object, code to create, CD:1825
- `Add Windows PrinterConnection()` (WshNetwork object), CD:1964
- `add()`, CD:1659-1661, CD:1909
- `Add()` (Dictionary object), CD:1825-1826
- `addEventListener()`, CD:1744-1746
- `AddHeader()`, code, 1335
- `AddPrinterConnection()`, CD:1964-1965
- `AddWindowsPrinterConnection()`, code, CD:1966
- `alert()`, 103-106, 1085, 1090
- Anchor object, 564
- `anchor()` (String object), 498
- `Anchor.unwatch()`, 568-569
- `Anchor.watch()`, 569-570
- `AppActivate()`, CD:1990-1991
- `appendChild()`, CD:1455-1457
- `appendData()`, CD:1407, CD:1410
- `AppendToLog()`, 1335-1336
- Applet object, code to access, 572
- Application object, 1178
- `Application.Contents.Lock()`, 1181-1182
- `Application.Contents.Remove()`, 1180
- `Application.Contents.RemoveAll()`, 1180-1181
- `Application.Unlock`, 1182-1183
- `apply()`, 353-354
- Area object, 573
- `Area.handleEvent()`, 576-577
- `Area.unwatch()`, 594
- `Area.watch()`, 596
- Array object, 42, 250-251, 254-272
- `as()`, code, 409-410
- `asin()`, 408, 411-412
- assigning, code, 258-259, 275-276
- `atan()`, 408, 412-413
- `atan2()`, 408, 413-414
- `atEnd()`, 337-338
- `atob()`, 1085, 1091
- `back()`, 799-800, 1085, 1092, 1117
- `beginTransaction()`, 1201-1202, 1234-1236, 1199
- `big()` (String object), 498
- `BinaryRead()`, 1318-1319
- `BinaryWrite()`, 1335-1336
- `blink()` (String object), 498
- blob object, 1189
- `blobImage()`, 1189-1191
- `blobLink()`, 1191-1192
- `blur()`
 - Button object, 598
 - buttons, code to remove focus, 599-600
 - check boxes, code to remove focus, 617-618
 - Checkbox object, 615
 - code, 747, 780, CD:1482, CD:1589, CD:1661, CD:1716
 - FileUpload object, 746
 - Frame object, 779
 - HTMLAnchorElement object, CD:1481
 - HTMLInputElement object, CD:1586
 - HTMLSelectElement object, CD:1659
 - HTMLTextAreaElement object, CD:1715
 - Password object, 922
 - Radio object, 938
 - Reset object, 952
 - select boxes, code to remove focus, 977
 - Select object, 975
 - Submit button, code to remove focus, 1032
 - Submit object, 1030
 - text areas, code to remove focus, 1065-1066
 - text boxes, code to remove focus, 1048
 - Text object, 1046
 - Textarea object, 1063
 - window focus, code to remove, 1092-1093
 - Window object, 1085
- `bold()` (String object), 498
- Boolean object, 274
 - `toSource()`, 276-277
 - `toString()`, 277-278
 - `unwatch()`, 278
 - `valueOf()`, 279
 - `watch()`, 279-280
- `borderWidths()`, 650, 663, 689, 995, 1005
- `btoa()`, 1085, 1094
- `BuildPath()` (FileSystemObject object), CD:1864-1866

- Button object, 598
 - blur(), 599
 - click(), 600
 - focus(), 601
 - handleEvent(), 603
 - unwatch(), 610
 - watch(), 613
- byteToString(), 1275-1278
- call(), 353, 357, 397-399
- captureEvents(), 647-648, 827-828
 - Document object, 637
 - Layer object, 822
 - window.document property, 1108
 - Window object, 1085
- ceil(), 414-415
- CharacterData object, CD:1407
 - appendData(), CD:1409-1410
 - deleteData(), CD:1411
 - insertData(), CD:1410
 - replicateData(), CD:1411
 - substringData(), CD:1409
- charAt() (String object), 498
- charCodeAt() (String object), 498
- Checkbox object, 615
 - blur(), 617
 - click(), 620
 - focus(), 622
 - handleEvent(), 624
 - unwatch(), 632
 - watch(), 635
- Clear(), 1335, 1339
- clearError(), 1275, 1278-1279
- clearInterval(), 779-781, 1085, 1097
- clearTimeout(), 779-781, 1085, 1098
- click()
 - Button object, 598
 - Checkbox object, 615
 - code, 942, 954-955, 600-601, 620-621, CD:1590-1591
 - forms, code to submit, 1033
 - HTMLInputElement object, CD:1586
 - Radio object, 938
 - Reset object, 952
 - Submit object, 1030
- client.destroy(), 1195
- client.expiration(), 1196
- client.unwatch(), 1197
- client.watch(), 1198
- cloneNode(), CD:1455, CD:1458
- close()
 - code, 1099-1100, CD:1529, CD:1915
 - Cursor object, 1219
 - cursors, code to close, 1221-1222, 1389-1390
 - Document object, 637
 - document output streams, code to close, 651-652
 - File object, 1275
 - files, code to close, 1279
 - HTMLDocument object, CD:1525
 - ResultSet object, 1345
 - resultsets, code to close, 1346-1347
 - Stproc object, 1388
 - TextStream object, CD:1912
 - Window object, 1085
 - window.document property, 1108
- columnName()
 - code, 1347-1348
 - Cursor object, 1220
 - cursors, code to close, 1223-1224
 - ResultSet object, 1345
- columns()
 - code, 1224, 1348-1349
 - Cursor object, 1220
 - ResultSet object, 1345
- commitTransaction(), 1199, 1202-1203, 1234, 1237
- compile(), 467, 476-477
- concat()
 - arg1,;argN argument, 251
 - arguments, 251
 - Array object, 42, 250-251
 - grocery inventory, code to display, 251-252
 - return values, 251
 - String object, 498
- confirm(), 1085, 1103
- connect()
 - code, 1238-1239
 - commitFlag parameter, 1238, 1259
 - database object, 1234
 - databases, code to connect to, 1259-1260
 - databases, code to disconnect from, 1263-1264
 - dbInstance parameter, 1238, 1258
 - dbName parameter, 1238, 1259

- DbPool object, 1257
- dbType parameter, 1238, 1258
- maxConn parameter, 1238, 1259
- parameters, 1238, 1258
- pwd parameter, 1238, 1258
- uid parameter, 1238, 1258
- connected()
 - Connection object, 1199
 - connections, code to test,
1203-1204, 1239-1240, 1260
 - database object, 1234
 - DbPool object, 1257
- connection(), 1257, 1261
- Connection object, 1199-1200
 - beginTransaction(), 1201
 - commitTransaction(), 1202
 - connected(), 1203
 - connection(), 1234
 - cursor(), 1204
 - execute(), 1205
 - majorErrorCode(), 1205
 - majorErrorMessage(), 1206
 - minorErrorCode(), 1207
 - minorErrorMessage(), 1208
 - release(), 1210
 - rollbackTransaction(), 1211
 - SQLTable(), 1212
 - storedProc(), 1213-1214
 - toString(), 1215
 - unwatch(), 1216
 - watch(), 1218
- ConnectObject(), CD:1929, CD:1932
- Contents.Lock() (Application object),
1178
- Contents.Remove(), 1178 1379
- Contents.RemoveAll(), 1178, 1379
- Contents.Unlock() (Application object),
1178
- contextual(), 637, 652
- Copy()
 - code, CD:1894
 - File object, CD:1845,
CD:1847-1848
 - Folder object, CD:1891
- CopyFile() (FileSystemObject object),
CD:1864, CD:1866-1867
- CopyFolder() (FileSystemObject
object), CD:1864, CD:1867-1868
- cos(), 408, 415-416
- Count()
 - arguments, code to retrieve,
CD:1961
 - code, CD:1946, CD:2014
 - environment variables, code to
display, CD:1956
 - unnamed parameters, code to dis-
play, CD:2017-2018
 - WshArguments object, CD:1944
 - WshEnvironment object,
CD:1954
 - WshNamed object, CD:1960
 - WshSpecialFolders object,
CD:2013
 - WshUnnamed object, CD:2016
- createAttribute() (Document object),
CD:1413-1414
- createAttributeNS() (Document object),
CD:1413
- createCaption(), CD:1689, CD:1693
- createCDATASection() (Document
object), CD:1413
- createComment() (Document object),
CD:1413
- createCSSStyleSheet(), CD:1810-1811
- createDocument(), CD:143-1432
- createDocumentFragment(), CD:1413,
CD:1416-1417, CD:1425
- createDocumentType(), CD:1431-1433
- createElement(), 149, CD:1413
- createElementNS() (Document object),
CD:1413
- createEntityReference() (Document
object), CD:1413
- createEvent(), CD:1733-1734
- DocumentEvent object, createEvent()
method, CD:1731-1732
- CreateFolder() (FileSystemObject
object), CD:1864, CD:1869
- createHTMLDocument(),
CD:1537-1538
- CreateObject()
 - code, 1369, CD:1933
 - remote scripts, code to create,
CD:1953-1954
 - Server object, 1368
 - WScript object, CD:1929
- createProcessingInstruction()
(Document object), CD:1413
- CreateScript(), CD:1953
- CreateShortcut(), CD:1990-1992

- CreateTextfile() (FileSystemObject object), CD:1864, CD:1869
 - code, CD:1870
 - description, CD:1870
 - filename parameter, CD:1870
 - overwrite parameter, CD:1870
 - unicode parameter, CD:1870
- createTextNode(), CD:1413, CD:1420
- createTFoot(), CD:1689, CD:1694
- createTHead(), CD:1689, CD:1694
- crypto.random, 1085, 1104
- crypto.signText, 1085, 1105
- CSSMediaRule object, CD:1775
 - deleteRule(), CD:1776
 - insertRule(), CD:1776-1777
- CSSPrimitiveValue object, CD:1781
 - getCounterValue(), CD:1782
 - getFloatValue(), CD:1782
 - getRectValue(), CD:1783
 - getRGBColorValue(), CD:1783-1784
 - getStringValue(), CD:1784
 - setFloatValue(), CD:1785
 - setStringValue(), CD:1786
- CSSRuleList object, CD:1790
- CSSRuleList.item(), CD:1791
- CSSStyleDeclaration object, CD:1792
 - getPropertyCSSValue(), CD:1794
 - getPropertyPriority(), CD:1794-1795
 - getPropertyValue(), CD:1795
 - item(), CD:1796
 - removeProperty(), CD:1797
 - setProperty(), CD:1798
- CSSStyleSheet object, CD:1801
 - deleteRule(), CD:1802
 - insertRule(), CD:1802-1803
- CSSValueList object, item(), CD:1807
- Cursor object, 1219
- cursor()
 - code, 1204
 - Connection object, 1199
 - database object, 1234
 - databases, code to run queries, 1240
- Cursor.close(), 1221
- Cursor.columnName(), 1222-1223
- Cursor.columns(), 1224
- Cursor.deleteRow(), 1225
- Cursor.insertRow(), 1226
- Cursor.next(), 1227
- Cursor.unwatch(), 1229
- Cursor.updateRow(), 1231
- Cursor.watch(), 1232
- database object, 1234
- database.beginTransaction(), 1235
- database.commitTransaction(), 1236
- database.connect(), 1237-1238
- database.connected(), 1239
- database.cursor(), 1240
- database.disconnect(), 1241
- database.execute(), 1242
- database.majorErrorCode(), 1242-1243
- database.majorErrorMessage(), 1243
- database.minorErrorCode(), 1244
- database.minorErrorMessage(), 1245
- database.rollbackTransaction(), 1247
- database.SQLTable(), 1248
- database.storedProc(), 1249
- database.storedProcArgs(), 1250
- database.toString(), 1251
- database.unwatch(), 1253
- database.watch(), 1254
- Date object, 286-287
 - getDate(), 289-290
 - getDay(), 290-291
 - getFullYear(), 291-292, 305
 - getHours(), 292
 - getMilliseconds(), 292-293
 - getMinutes(), 293
 - getMonth(), 294-295
 - getSeconds(), 295-296
 - getTime(), 296-297
 - getTimezoneOffset(), 297
 - getUTCDate(), 298
 - getUTCDay(), 298-299
 - getUTCFullYear(), 300
 - getUTCHours(), 300-301
 - getUTCMilliseconds(), 301
 - getUTCMinutes(), 301-302
 - getUTCMonth(), 302-304
 - getUTCSeconds(), 304
 - getYear(), 305-306
 - parse(), 306
 - setDate(), 308-310
 - setHours(), 310
 - setMilliseconds(), 310-311
 - setMinutes(), 311-312
 - setMonth(), 312-313
 - setSeconds(), 313-314
 - setTime(), 314
 - setUTCDate(), 314-315

- setUTCFullYear(), 315-316
- setUTCHours(), 316-317
- setUTCMilliseconds(), 317
- setUTCMinutes(), 318
- setUTCMonth(), 318-319
- setUTCSeconds(), 319-320
- setYear(), 320-321
- toGMTString(), 321
- toLocaleString(), 322
- toSource(), 322
- toString(), 323-324
- unwatch(), 325
- UTC(), 326
- valueOf(), 327
- watch(), 328
- DbPool object, 1257
- DbPool(), 1262-1263
- DbPool.connect(), 1258
- DbPool.connected(), 1260
- DbPool.connection(), 1199, 1261
- DbPool.DbPool(), 1262
- DbPool.disconnect(), 1263
- DbPool.majorErrorCode(), 1264
- DbPool.majorErrorMessage(), 1264-1265
- DbPool.minorError Message(), 1266
- DbPool.minorErrorCode(), 1265
- DbPool.storedProcArgs(), 1268
- DbPool.toString(), 1269
- DbPool.unwatch(), 1270
- DbPool.watch(), 1272
- debug(), 167
- decodeURI() (Global object), 365
- decodeURIComponent() (Global object), 365
- Delete(), CD:1845, CD:1851-1852, CD:1891, CD:1897
- deleteCaption(), CD:, CD:1689
- deleteCell(), code, CD:1707
- deleteData(), CD:1407, CD:1411, CD:1703
- DeleteFile() (FileSystemObject object), CD:1864, CD:1871
- DeleteFolder() (FileSystemObject object), CD:1864, CD:1872
- deleteRow()
 - code, 1225-1226, CD:1695-1696, CD:1712
 - Cursor object, 1220
- HTMLTableElement object, CD:1689
- HTMLTableSectionElement object, CD:1710
- deleteRule(), CD:1775-1776, CD:1801-1802
- deleteTFoot(), CD:1689, CD:1696
- deleteTHead(), CD:1689, CD:1697
- destroy(), 1193-1195
- Dictionary object, CD:1824
- dimensions (VBAArray object), 551
- disableExternalCapture, 1085, 1106-1107
- disconnect(), 1234, 1241-1242, 1257
- DisconnectObject() (WScript object), CD:1929
- dispatchEvent() (EventTarget object), CD:1744-1746
- Document object, 637, CD:1413
- document.all property array, 639
- document.all.item(), 639
- document.all.tags(), 640
- document.captureEvents(), 646
- document.close(), 651
- document.contextual(), 652
- Document.createAttribute(), CD:1414
- Document.createAttributeNS(), CD:1414
- Document.createCDATASection(), CD:1415
- Document.createComment(), CD:1416
- Document.createDocumentFragment(), CD:1416
- Document.createElement(), CD:1417
- Document.createElementNS(), CD:1417-1418
- Document.createEntityReference(), CD:1418
- Document.createProcessingInstruction(), CD:1419
- Document.createTextNode(), CD:1419-1420
- Document.getElementById(), CD:1421
- Document.getElementsByTagName(), CD:1422
- Document.getElementsByTagNameNS(), CD:1422-1423
- document.getSelection(), 659
- document.handleEvent(), 660
- Document.importNode(), CD:1424
- document.open(), 680

- document.releaseEvents(), 683
- document.routeEvent(), 685
- document.unwatch(), 690
- document.watch(), 693
- document.write(), 694
- document.writeln(), 695
- DocumentCSSList object, CD:1809
- DocumentCSSList.getOverrideStyle(), CD:1809
- DOMImplementation object, CD:1431
- DOMImplementation.createDocument(), CD:1432
- DOMImplementation.createDocumentType(), CD:1433
- DOMImplementation.hasFeature(), CD:1433
- DOMImplementationCSS object, CD:1810
- DOMImplementationCSS.createCSSStyleSheet(), CD:1810-1811
- DriveExists() (FileSystemObject object), CD:1865, CD:1873
- Echo(), CD:1929, CD:1934
- Element object, CD:1434
- Element.getAttribute(), CD:1435
- Element.getAttributeNode(), CD:1436
- Element.getAttributeNodeNS(), CD:1436-1437
- Element.getAttributeNS(), CD:1437
- Element.getElementsByTagName(), CD:1438
- Element.getElementsByTagNameNS(), CD:1438
- Element.hasAttribute(), CD:1439
- Element.hasAttributeNS(), CD:1439
- Element.removeAttribute(), CD:1440
- Element.removeAttributeNode(), CD:1440-1441
- Element.removeAttributeNS(), CD:1441
- Element.setAttribute(), CD:1442
- Element.setAttributeNode(), CD:1442
- Element.setAttributeNodeNS(), CD:1443
- Element.setAttributeNS(), CD:1444
- enableExternalCapture, 1085, 1111
- enablePrivilege(), 22
- encodeURI() (Global object), 365
- encodeURIComponent() (Global object), 366
- encryption, crypto property, code, 1103-1104
- End(), 1335, 1340
- Enumerator object, 337
- Enumerator.atEnd(), 337
- Enumerator.item(), 338
- Enumerator.moveFirst(), 339
- Enumerator.moveNext(), 340
- EnumNetworkDrives(), CD:1964, CD:1967
- EnumPrinterConnection(), CD:1964, CD:1968
- eof() (File object), 1275 of, 1280-1281
- equals(), 397, 400
- error(), 1275, 1281-1282
- errorCode(), 1355, 1359-1360
- errorMessage() (SendMail object), 1355
- escape(), 366, 371
- eval(), 366, 397, 400-401, 452-454
- Event object, 698, CD:1733
- Event.initEvent(), CD:1737
- Event.preventDefault(), CD:1738
- Event.stopPropagation(), CD:1739
- event.unwatch(), 715-716
- event.watch(), 716
- EventTarget object, CD:1744
- EventTarget.addEventListener(), CD:1745
- EventTarget.dispatchEvent(), CD:1746
- EventTarget.removeEventListener(), CD:1747
- exec(), 467, 477-478
- Exec(), CD:1990, CD:1994
- exec(str), 89
- Execute(), 1368-1370, CD:1972, CD:1976
- execute(), 1199, 1205, 1234, 1242
- Exists() (Dictionary object), CD:1825-1827, CD:1960
- exists(), 1275, 1282
- exp(), 408, 417-418
- ExpandEnvironmentStrings(), CD:1990, CD:1994-1995
- expiration(), 1193, 1196
- ExtensionName() (FileSystemObject object), CD:1865
- File object, 1275, CD:1845
- File.byteToString(), 1277
- File.clearError(), 1278
- File.close(), 1279
- File.eof(), 1280

File.error(), 1281
File.exists(), 1282
File.flush(), 1283
File.getLength(), 1284
File.getPosition(), 1284-1285
File.open(), 1285
File.read(), 1288
File.readByte(), 1288-1289
File.readLine(), 1289
File.setPosition(), 1290
File.stringToByte(), 1291
File.unwatch(), 1292
File.watch(), 1294
File.write(), 1295
File.writeByte(), 1296
File.writeln(), 1297
FileExists() (FileSystemObject object),
 CD:1865, CD:1874-1875
FileUpload object, 746
FileUpload.blur(), 747
FileUpload.focus(), 748
FileUpload.handleEvent(), 750-751
FileUpload.select(), 756
FileUpload.unwatch(), 758
FileUpload.watch(), 760
find(), 1085, 1112
fixed() (String object), 498
floor(), 408, 418-419
Flush(), 1335, 1342
flush(), 1275, 1283
focus()
 Button object, 598
 buttons, code to shift focus,
 601-602
 check boxes, code to apply focus,
 623
 Checkbox object, 615
 code, 782, 925, 955-956,
 1112-1113, CD:1484
 FileUpload object, 746
 Frame object, 779
 HTMLAnchorElement object,
 CD:1481
 HTMLInputElement object,
 CD:1586
 HTMLSelectElement object,
 CD:1659
 HTMLTextAreaElement object,
 CD:1715
 Password object, 923
 Radio object, 939
 Reset object, 952
 select boxes, code to place cursor,
 978-979
 Select object, 975
 Submit button, code to set focus,
 1034
 Submit object, 1030
 text areas, code to place cursor,
 1067-1068
 text boxes, code to place cursor,
 1050-1051
 Text object, 1046
 Textarea object, 1063
 Window object, 1086
Folder object, CD:1891
Folder.Copy(), CD:1893-1894
Folder.Delete(), CD:1897
Folder.Move(), CD:1900
FolderExists() (FileSystemObject
 object), CD:1865, CD:1876-1877
Folders.Add(), CD:1909
fontcolor() (String object), 498
fontsize() (String object), 498
Form object, 761
Form.handleEvent(), 767
Form.reset(), 773
Form.submit(), 774
Form.unwatch(), 776
Form.watch(), 777
forward(), 799, 801-802, 1086, 1114,
 1117, 1171
Frame object, 779
Frame.blur(), 779-780
Frame.clearInterval(), 780
Frame.clearTimeout(), 781
Frame.focus(), 782
Frame.print(), 787
Frame.setInterval(), 788-789
Frame.setTimeout(), 789
Frame.unwatch(), 790-791
Frame.watch(), 791
fromCharCode() (String object), 498
Function object, 353
Function.apply(), 353-354
Function.call(), 356
Function.toSource(), 361-362
Function.toString(), 362
Function.valueOf(), 363
GET, 124-125

- GetAbsolutePathName()
(FileSystemObject object), CD:1865, CD:1877
- getAttribute(), CD:1434-1436, CD:1443
- getAttributeNode(), CD:1434-1436
- getAttributeNodeNS(), CD:1434, CD:1437
- getAttributeNS(), CD:1434, CD:1437
- GetBaseName() (FileSystemObject object), CD:1865, CD:1878
- getComputedStyle(), CD:1818-1819
- getCounterValue(), CD:1781-1782
- getDate() (Date object), 286, 289-290
- getDay() (Date object), 286, 290-291
- GetDrive() (FileSystemObject object), CD:1833-1834, CD:1865, CD:1879
- GetDriveName() (FileSystemObject object), CD:1865, CD:1879-1880
- getElementById(), 149, 152, CD:1413
- getElementsByName(), CD:1525, CD:1531-1532
- getElementsByTagName(), CD:1413, CD:1434, CD:1438
- getElementsByTagNameNS(), CD:1413, CD:1434, CD:1438-1439
- GetExtensionName()
(FileSystemObject object), CD:1880-1881
- GetFile() (FileSystemObject object), CD:1865, CD:1882
- GetFileName() (FileSystemObject object), CD:1865, CD:1882-1883
- getFloatValue(), CD:1781-1783
- GetFolder() (FileSystemObject object), CD:1865, CD:1883-1884
- getFullYear() (Date object), 286, 291-292, 305
- getHours() (Date object), 286, 292
- getItem (VBAArray object), 551
- GetLastError(), 1368-1370
- getLength(), 1275, 1284
- getMember(), 397, 402
- getMilliseconds() (Date object), 286, 292-293
- getMinutes() (Date object), 287, 293
- getMonth() (Date object), 287, 294-295
- getNamedItem(), CD:1448-1449
- getNamedItemNS(), CD:1449
- GetObject(), CD:1929, CD:1935
- getOverrideStyle(), CD:1809-1810
- GetParentFolderName()
(FileSystemObject object), CD:1865, CD:1885
- getPosition(), 1275, 1285
- getPropertyCSSValue(), CD:1792-1794
- getPriorityPriority(), CD:1792, CD:1795
- getPropertyValue(), CD:1793-1796
- getRectValue(), CD:1781-1783
- getRGBColor(), CD:1781-1784
- getSeconds() (Date object), 287, 295-296
- getSelection(), 637, 1108
- getSlot(), 397, 403
- GetSpecialFolder() (FileSystemObject object), CD:1865, CD:1885-1886
- getStringValue(), CD:1781, CD:1784
- GetTempName() (FileSystemObject object), CD:1865, CD:1886-1887
- getTime() (Date object), 287, 296-297
- getTimezoneOffset() (Date object), 287, 297
- getUTCDate() (Date object), 287, 298
- getUTCDay() (Date object), 287, 298-299
- getUTCFullYear() (Date object), 287, 300
- getUTCHours() (Date object), 287, 300-301
- getUTCMilliseconds() (Date object), 287, 301
- getUTCMinutes() (Date object), 287, 301-302
- getUTCMonth() (Date object), 287, 302-304
- getUTCSeconds() (Date object), 287, 304
- getVarDate() (Date object), 287, 304-305
- getWindow(), 397, 403
- getWrappedException(), 395-396
- getYear() (Date object), 287, 305-306
- Global object, 365
- Global.escape(), 370
- Global.isFinite(), 374
- Global.Number(), 376-377
- Global.parseFloat, 377
- Global.parseInt, 378
- Global.String, 379
- Global.unescape, 380
- go(), 799, 802, 1117

- handleEvent()
 - Area object, 573
 - Button object, 598
 - Checkbox object, 615
 - clicks on pages, code, 1052-1053, 1070-1071
 - code, 603-605, 751-752, 926-927, 945, 957-958
 - events, code to pass, 625, 660-661
 - FileUpload object, 746
 - Form object, 761
 - Image object, 808
 - Layer object, 822
 - Password object, 923
 - Radio object, 939
 - Reset object, 952
 - Select object, 975
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1063
 - Window object, 1086
 - window.document property, 1108
- hasAttribute(), CD:1434, CD:1439
- hasAttributeNS(), CD:1434, CD:1440
- hasAttributes(), CD:1455, CD:1459-1460
- hasChildNodes() (Node object), CD:1455
- hasChildren(), code, CD:1417
- hasFeature(), 149, CD:1431, CD:1434
- Hidden object, 793
- Hidden.unwatch(), 796
- Hidden.watch(), 798
- History object, 799
- History.back(), 800
- History.forward(), 801
- History.go(), 802
- History.unwatch(), 805
- History.watch(), 806
- home, 1086, 1118
- HTMLAnchorElement object, CD:1481
- HTMLAnchorElement.blur(), CD:1482
- HTMLAnchorElement.focus(), CD:1484
- HTMLAreaElement object, code to create, CD:1498
- HTMLBaseElement object, code to create, CD:1503
- HTMLBaseFontElement object, code to create, CD:1505
- HTMLBodyElement object, code to create, CD:1508
- HTMLBRElement object, code to create, CD:1512
- HTMLButtonElement object, code to create, CD:1514
- HTMLCollection object, CD:1518
- HTMLCollection.item(), CD:1519
- HTMLCollection.namedItem(), CD:1520-1521
- HTMLDocument object, CD:1525
- HTMLDocument.close(), CD:1528
- HTMLDocument.getElementsByName(), CD:1531
- HTMLDocument.open(), CD:1533
- HTMLDocument.write(), CD:1536
- HTMLDocument.writeln(), CD:1537
- HTMLDOMImplementation object, CD:1537
- HTMLDOMImplementation.createHTMLDocument(), CD:1538
- HTMLEncode(), 1368-1370
- HTMLFormElement.reset(), CD:1552
- HTMLFormElement.submit(), CD:1553
- HTMLInputElement object, CD:1586
- HTMLInputElement.blur(), CD:1589
- HTMLInputElement.click(), CD:1590
- HTMLInputElement.focus(), CD:1593
- HTMLInputElement.select(), CD:1596
- HTMLSelectElement object, CD:1659
- HTMLSelectElement.add(), CD:1660
- HTMLSelectElement.blur(), CD:1661
- HTMLSelectElement.focus(), CD:1662
- HTMLSelectElement.remove(), CD:1666
- HTMLTableElement object, CD:1689
- HTMLTableElement.createCaption(), CD:1693
- HTMLTableElement.createTFoot(), CD:1693-1694
- HTMLTableElement.createTHead(), CD:1694
- HTMLTableElement.deleteCaption(), CD:1695
- HTMLTableElement.deleteRow(), CD:1695
- HTMLTableElement.deleteTFoot(), CD:1696
- HTMLTableElement.deleteTHead(), CD:1696

- HTMLTableElement.insertRow(),
CD:1698
- HTMLTableRowElement object,
CD:1703
- HTMLTableRowElement.deleteCell(),
CD:1706
- HTMLTableRowElement.insertCell(),
CD:1707
- HTMLTableSectionElement object,
CD:1710
- HTMLTableSectionElement.deleteRow(),
CD:1712
- HTMLTableSectionElement.deleteSection(),
CD:1712
- HTMLTableSectionElement.insertRow(),
CD:1713
- HTMLTableSectionElement.insertSection(),
CD:1713
- HTMLTextAreaElement object,
CD:1715
- HTMLTextAreaElement.blur(),
CD:1716
- HTMLTextAreaElement.focus(),
CD:1718-1719
- HTMLTextAreaElement.select(),
CD:1721
- Image object, 808
- Image.handleEvent(), 811
- Image.unwatch(), 818
- Image.watch(), 819
- importNode() (Document object),
CD:1413
- indexOf(), 177, 498
- initEvent(), CD:1733, CD:1737-1738
- initMouseEvent() (Event object),
CD:1748, CD:1756
- initUIEvent() (UIEvent object),
CD:1761
- insertBefore(), CD:1455, CD:1461
- insertCell(), CD:1703, CD:1707
- insertData(), CD:1407, CD:1410
- insertRow()
code, 1226-1227, CD:1698,
CD:1713
Cursor object, 1220
HTMLTableElement object,
CD:1689
HTMLTableSectionElement
object, CD:1710
- insertRule(), CD:1775-1777,
CD:1801-1803
- isFinite(), 366, 374-375, 387-388
- isNaN() (Global object), 366
- isSupported(), CD:1456, CD:1461-1462
- isValid(), 1301-1302
- italics() (String object), 498
- item(), CD:1450
code to invoke, CD:1796,
CD:1807-1808
code to iterate, CD:1791
CSSRuleList object, CD:1790
CSSStyleDeclaration object,
CD:1792
CSSValueList object, CD:1807
document.all property array, 639
drives, code to access, 338-339
Enumerator object, 337
HTML elements, code to find,
640
HTMLCollection object,
CD:1518
- Item()
system directory path, code to
display, CD:1957
WScript parameters, code to
display, CD:1946-1947
WshArguments object, CD:1944
WshEnvironment object,
CD:1954
- Items() (Dictionary object), CD:1825,
CD:1828-1829
- Java, code to access, 121
- javaArray.toString(), 392
- javaEnabled(), 904, 907-908
- join() (Array object), 42, 250, 254-255
- JSException.getWrappedException(),
396
- JSObject.call(), 398
- JSObject.equals(), 399-400
- JSObject.eval(), 400
- JSObject.getMember(), 401-402
- JSObject.getSlot(), 403
- JSObject.getWindow(), 403
- JSObject.removeMember(), 404
- JSObject.setMember(), 405
- JSObject.setSlot(), 405
- JSObject.toString(), 406
- Keys() (Dictionary object), CD:1825,
CD:1830-1831
- lastIndexOf() (String object), 498
- Layer object, 822
- Layer.captureEvents(), 827

- Layer.handleEvent(), 835
- Layer.load(), 838
- Layer.moveAbove(), 838
- Layer.moveBelow(), 839
- Layer.moveBy(), 840
- Layer.moveTo(), 841
- Layer.moveToAbsolute(), 842
- Layer.releaseEvents(), 852
- Layer.resizeBy(), 853-854
- Layer.resizeTo(), 854
- Layer.routeEvent(), 855
- Layer.unwatch(), 860
- Layer.watch(), 862
- lbound (VBAArray object), 551
- length (toSring() object), 390
- Link object, 865
- link() (String object), 498
- Link.handleEvent(), 866
- Link.unwatch(), 883
- Link.watch(), 884
- load(), 822, 838
- Location object, 886
- Location.reload(), 894
- Location.replace(), 895
- Location.unwatch(), 896
- Location.watch(), 898
- lock(), code, 1301-1303, 1312-1313, 1373-1375
- Lock() object, 1301
- Lock.isValid(), 1302
- Lock.lock(), 1303
- Lock.unlock(), 1305
- Lock.unwatch(), 1305
- Lock.watch(), 1307
- log(), 408, 421
- LogEvent(), CD:1990, CD:1995-1996
- majorErrorCode()
 - code, 1205-1206
 - Connection object, 1199
 - database connection errors, code to retrieve, 1243, 1264
 - database object, 1234
 - DbPool object, 1257
- majorErrorMessage()
 - code, 1206-1207
 - Connection object, 1199
 - database connection errors, code to retrieve, 1243-1244, 1265
 - database object, 1234
 - DbPool object, 1257
- MapNetworkDrive(), CD:1964, CD:1968-1969
- MapPath(), 1368, 1371
- margins(), 650, 663, 689, 995, 1017
- match(), 88, 499
- Math object, 46, 408-409
- Math.abs(), 46, 409
- Math.acos(), 46, 410
- Math.asin(), 46, 411
- Math.atan(), 46, 412
- Math.atan2(), 46, 413
- Math.ceil(), 46, 414
- Math.cos(), 46, 415
- Math.exp(), 46, 417
- Math.floor(), 46, 418
- Math.log(), 46, 420, 421
- Math.max(), 46, 423
- Math.min(), 46, 424
- Math.pow(), 46, 426
- Math.random(), 46, 427
- Math.round(), 46, 428
- Math.sin(), 46, 429
- Math.sqrt(), 46, 430
- Math.tan(), 46, 432
- Math.toSource(), 433
- Math.toString(), 434
- Math.unwatch(), 435
- Math.watch(), 435-436
- max(), 409, 423-424
- MimeType object, 899
- MimeType.unwatch(), 903
- MimeType.watch(), 903
- min(), 409, 424-425
- minorErrorCode()
 - code, 1207-1208
 - Connection object, 1199
 - database connection errors, code to retrieve, 1244-1245, 1265-1266
 - database object, 1234
 - DbPool object, 1257
- minorErrorMessage(), 1266-1267
 - code, 1208
 - Connection object, 1199
 - database connection errors, code to retrieve, 1245
 - database object, 1234
 - DbPool object, 1257
 - description, CD:1752
 - MouseEvent.initMouseEvent, CD:1752
 - syntax, CD:1752

- Move() (File object), CD:1845, CD:1853, CD:1891, CD:1900-1901
- moveAbove(), 822, 839
- moveBelow, 822, 840
- moveBy (Window object), 1086
- moveBy(), 822, 840-841, 1125-1126
- MoveFile() (FileSystemObject object), CD:1865, CD:1887-1888
- moveFirst(), 337-340
- MoveFolder() (FileSystemObject object), CD:1865, CD:1888-1889
- moveNext(), 337, 340-341
- moveTo(), 822, 841-842, 1086, 1127
- moveToAbsolute(), 822, 842-843
- MutationEvent.initMutationEvent(), CD:1759
- namedItem(), CD:1518, CD:1521
- NamedNodeMap.getNamedItem(), CD:1448
- NamedNodeMap.getNamedItemNS(), CD:1449
- NamedNodeMap.item(), CD:1449-1450
- NamedNodeMap.removeNamedItem(), CD:1451
- NamedNodeMap.removeNamedItemNS(), CD:1451
- NamedNodeMap.setNamedItem(), CD:1452
- NamedNodeMap.setNamedItemNS(), CD:1453
- navigator object, 904
- navigator.javaEnabled(), 907
- navigator.plugins.refresh(), 911
- navigator.preference(), 911
- navigator.taintEnabled(), 912
- navigator.unwatch(), 914
- navigator.watch(), 914
- next(), 1220, 1227-1228, 1345, 1349-1350
- Node object, CD:1455
- Node.appendChild(), CD:1457
- Node.cloneNode(), CD:1458
- Node.hasAttributes(), CD:1459
- Node.insertBefore(), CD:1460
- Node.isSupported(), CD:1461
- Node.normalize(), CD:1466
- Node.removeChild(), CD:1469
- Node.replaceChild(), CD:1469-1470
- normalize() CD:1456, CD:1466
- Number object, 440
- Number(), 377, 438-439
- Number.toExponential(), 446
- Number.toFixed(), 447
- Number.toLocaleString(), 447
- Number.toPrecision(), 448
- Number.toSource(), 448
- Number.toString(), 449
- Number.unwatch(), 449
- Number.valueOf(), 450-451
- Number.watch(), 451
- Object object, 452-453
- Object.eval(), 454
- Object.toSource(), 455
- Object.toString(), 456
- Object.unwatch(), 457
- Object.valueOf(), 458
- Object.watch(), 458
- ObjectContext object, 1309
- ObjectContext.SetAbort(), 1310-1311
- ObjectContext.SetComplete(), 1311
- onSubmit(), code, 772-773
- open(), 141, 175
 - a option, 1275, 1286
 - a+ option, 1275, 1286
 - alwaysLowered parameter, 1134
 - alwaysRaised parameter, 1134
 - code, 1136-1137, CD:1534
 - dependent parameter, 1134
 - directories parameter, 1134
 - Document object, 637
 - documents, code to open, 680
 - File object, 1275
 - files, code to open, 1286
 - height parameter, 1135
 - hotkeys parameter, 1135
 - HTMLDocument object, CD:1526
 - innerHeight parameter, 1135
 - innerWidth parameter, 1135
 - location parameter, 1135
 - menubar parameter, 1135
 - option option, 1276, 1286
 - options, 1275-1276, 1286
 - outerHeight parameter, 1135
 - outerWidth parameter, 1135
 - r option, 1275, 1286
 - r+ option, 1275, 1286
 - resizable parameter, 1135
 - screenX parameter, 1135
 - screenY parameter, 1135
 - scrollbars parameter, 1135
 - titlebar parameter, 1135

- toolbar parameter, 1135
- w option, 1276, 1286
- w+ option, 1276, 1286
- width parameter, 1135
- Window object, 1086
- window.document property, 1108
- z-lock parameter, 1136
- OpenAsTextStream() (File object),
CD:1845, CD:1854-1856
- OpenTextFile(), CD:1865,
CD:1889-1890
- Option object, 915
- Option.unwatch(), 919
- Option.watch(), 921
- outParamCount(), 1388-1391
- outParameters(), 1388, 1391-1392
- paddings(), 650, 663, 689, 995,
1021-1022
- parse() (Date object), 287, 306
- parseFloat(), 366, 377-378, 464, 1193
- parseInt(), 366, 378-379, 465, 1193
- Password object, 922
- Password.blur(), 923
- Password.focus(), 925
- Password.handleEvent(), 926
- Password.select(), 930
- Password.unwatch(), 931
- Password.watch(), 933
- pattern matching, 88-89
- Plugin object, 934
- Plugin.unwatch(), 937
- Plugin.watch(), 938
- plugins.refresh, 904, 911
- pop(), 42, 250, 257
- Popup(), CD:1990, CD:1996-1997
- POST, 124-125
- pow(), 409, 426-427
- preference(), 904, 912
- preventDefault(), CD:1733, CD:1738
- print(), 779, 788, 1086, 1146-1147
- project.lock(), 1313
- project.unlock(), 1313-1314
- project.unwatch(), 1314
- project.watch(), 1315
- prompt(), 1086, 1147
- push(), 42, 250, 259-260
- Quit(), CD:1929, CD:1937-1938
- Radio object, 938
- Radio.blur(), 939
- Radio.click(), 941
- Radio.focus(), 943
- Radio.handleEvent(), 944-945
- Radio.unwatch(), 949
- Radio.watch(), 951
- random(), 409, 427-428
- read(), 1275, 1288, CD:1917-1918
- Read() (TextStream object), CD:1912
- ReadAll(), CD:1912, CD:1918-1919
- readByte(), 1275, 1289
- ReadLine(), CD:1912, CD:1919
- readln(), 1275, 1290
- Redirect(), 1335, 1343
- RegDelete(), CD:1990, CD:1997-1998
- RegExp object, 467
- RegExp.compile(), 476
- RegExp.exec(), 477
- RegExp.test(), 487-488
- RegExp.toSource(), 488
- RegExp.toString(), 489
- RegExp.unwatch(), 490
- RegExp.valueOf(), 491
- RegExp.watch(), 492
- RegRead(), CD:1990, CD:1998
- RegWrite(), CD:1990, CD:1999
- release(), 1199, 1210-1211
- releaseEvents (Window object), 1086
 - code, 684-685, 852-853
 - Document object, 637
 - Layer object, 822
 - window.document property, 1108
- reload(), 886, 894-895
- Remove(), 1382, CD:1666-1667,
CD:1825, CD:1831-1832, CD:1954,
CD:1959
- remove() (HTMLSelectElement object),
CD:1659
- RemoveAll(), 1382-1383, CD:1825,
CD:1832
- removeAttribute(), CD:1435, CD:1440
- removeAttributeNode(), CD:1435,
CD:1441
- removeAttributeNS(), CD:1435,
CD:1441
- removeChild(), CD:1456, CD:1469
- removeEventListener(), CD:1744,
CD:1747
- removeMember(), 397, 404
- removeNamedItem(), code, CD:1451
- removeNamedItemNS(), code,
CD:1451-1452
- RemoveNetworkDrive(), CD:1964,
CD:1969

- RemovePrinterConnection(), CD:1964, CD:1970
- removeProperty(), CD:1793, CD:1798
- replace(), 499, 886, 895
- replace(reqExpObj,str), 88
- replaceChild(), CD:1456, CD:1470
- replaceData(), CD:1407
- replicateData(), code, CD:1411-1412
- Request object, 1318
- Request.BinaryRead(), 1318-1319
- Request.ReadBinary(), 1322
- request.unwatch(), 1333
- request.watch(), 1334
- Reset object, 952
- reset(), 761, 773, CD:1547, CD:1552-1553
- Reset.blur(), 953
- Reset.click(), 954
- Reset.focus(), 955
- Reset.handleEvent(), 957
- Reset.unwatch(), 962-963
- Reset.watch(), 965
- resizeBy(), 822, 854, 1086, 1150-1151
- resizeTo(), 822, 855, 1086, 1152
- Response object, 1335
- Response.AddHeader(), 1335
- Response.AppendToLog(), 1336
- Response.BinaryWrite(), 1336
- Response.Clear(), 1338
- Response.End(), 1340
- Response.Flush(), 1341-1342
- Response.Redirect(), 1343
- Response.Write(), 1344
- Resultset object, 1345
- resultSet(), 1388, 1394
- Resultset.close(), 1346
- Resultset.columnName(), 1347
- Resultset.columns(), 1348
- Resultset.next(), 1349
- Resultset.unwatch(), 1351
- Resultset.watch(), 1353
- returnValue(), 1388, 1395
- reverse() (Array object), 42, 250, 260-261
- rollbackTransaction(), 1199, 1211-1212, 1234, 1247
- round(), 409, 428-429
- routeEvent(), 637, 686-687, 822, 855-856, 1086, 1108
- Run(), CD:1990, CD:2000
- Save(), CD:2005, CD:2009-2010, CD:2020-2022
- screen.unwatch(), 972
- screen.watch(), 973
- scroll(), 1086, 1156-1157
- scrollBy, 1086, 1159-1160
- scrollTo(), 1086, 1160-1161
- search() (String object), 499
- search(reqExpObj), 88
- Select object, 975
- select()
 - code, 756-757, 930, CD:1596, CD:1722
 - FileUpload object, 746
 - HTMLInputElement object, CD:1586
 - HTMLTextAreaElement object, CD:1715
 - Password object, 923
 - text areas, code to select text, 1079-1080
 - text boxes, code to select text, 1059
 - Text object, 1046
 - Textarea object, 1063
- Select.blur(), 976-977
- Select.focus(), 978
- Select.handleEvent(), 980
- Select.unwatch(), 992
- Select.watch(), 993
- send(), 1355, 1364-1365
- SendKeys(), CD:1990, CD:2002
- SendMail object, 1355
- SendMail.errorCode(), 1359
- SendMail.errorMessage(), 1360-1361
- SendMail.send(), 1364
- SendMail.unwatch(), 1366
- SendMail.watch(), 1367
- Server object, 1368, 1373
- Server.CreateObject(), 1369
- Server.Execute(), 1369-1370
- Server.GetLastError(), 1370
- Server.HTMLEncode(), 1370
- server.lock(), 1375
- Server.MapPath(), 1371
- Server.Transfer(), 1372
- server.unlock(), 1376
- server.unwatch(), 1377
- Server.URLEncode(), 1372
- server.watch(), 1378
- Session object, 1379
- Session.Abandon(), 1379
- Session.Contents.Remove(), 1381

Session.Contents.RemoveAll(), 1382
SetAbort(), 1308, 1311
setAttribute(), CD:1435, CD:1442
setAttributeNode(), CD:1435,
 CD:1442-1443
setAttributeNodeNS(), CD:1435,
 CD:1443
setAttributeNS(), CD:1435, CD:1444
SetComplete(), 1309-1312
setDate() (Date object), 287, 308-309
SetDefaultPrinter(), CD:1964,
 CD:1970-1971
setFloatValue(), CD:1781,
 CD:1785-1786
setFullYear() (Date object), 287,
 309-310
setHotKeys(), 1086, 1162
setHours() (Date object), 287, 310
setInterval (Window object), 1086
setInterval(), 95-96, 779, 789, 1163
setMember(), 397, 405
setMilliseconds() (Date object), 287,
 310-311
setMinutes() (Date object), 287,
 311-312
setMonth() (Date object), 287, 312-313
setNamedItem(), code, CD:1452
setNamedItemNS(), code, CD:1453
setPosition(), 1275, 1290-1291
setProperty(), CD:1793, CD:1798
setResizable, 1086, 1164
setSeconds() (Date object), 287,
 313-314
setSlot(), 397, 406
setStringValue(), CD:1781, CD:1786
setTime() (Date object), 287, 314
setTimeout(), 96, 779, 789-790, 1086,
 1164-1165
setUTCDate() (Date object), 287,
 314-315
setUTCFullYear() (Date object), 287,
 315-316
setUTCHours() (Date object), 287,
 316-317
setUTCMinutes() (Date object), 287, 318
setUTCMonth() (Date object), 287,
 318-319
setUTCSeconds() (Date object), 288,
 319-320
setYear() (Date object), 288, 320-321
setZOption, 1086, 1166
shift(), 42, 250, 261-262
ShowUsage(), CD:1944, CD:1949-1950
sin(), 409, 429-430
Skip(), CD:1912, CD:1920
SkipLine(), CD:1912, CD:1921
Sleep(), CD:1929, CD:1939
slice(), 42, 250, 262-263, 499
small() (String object), 499
sort(), 42, 250, 263-265
splice(), 42, 250, 265-266
split() (String object), 499
split(regExpObj,max), 88
splitText() (Text object), CD:1474-1475
SQLTable(), 1199, 1213, 1234,
 1248-1249
sqrt(), 409, 430-431
stop(), 1087, 1168-1169
stopPropagation(), CD:1733, CD:1739
storedProc(), 1200, 1214-1215, 1234,
 1250
storedProcArgs(), 1234, 1251, 1257,
 1268-1269
Stproc object, 1388
Stproc.close(), 1389
Stproc.outParamCount(), 1390
Stproc.outParameters(), 1391
Stproc.resultSet(), 1393
Stproc.returnValue(), 1395
Stproc.unwatch(), 1395-1396
Stproc.watch(), 1397
 (str), 89
strike() (String object), 499
String object, 498-500
String(), code, 379
String.anchor(), 501, 553
String.big(), 502
String.blink(), 503
String.bold(), 503-504
String.charAt(), 504-505
String.charCodeAtAt(), 505-506
String.concat(), 507
String.fixed(), 508
String.fontcolor(), 509
String.fontSize(), 509-510
String.fromCharCode(), 510-511
String.indexOf(), 511-512
String italics(), 512-513
String.lastIndexOf(), 513-514
String.localeCompare(), 515-16
String.match(), 516-517
String.replace(), 519

- String.search(), 520-521
- String.slice(), 521
- String.small(), 522
- String.split(), 522-524
- String.strike(), 524-525
- String.sub(), 525
- String.substr(), 526
- String.substring(), 527-528
- String.sup(), 528
- String.toLocaleLowerCase(), 528-529
- String.toLocaleUpperCase(), 529-530
- String.toLowerCase(), 530-531
- String.toString(), 532
- String.toUpperCase(), 533
- String.unwatch(), 533-534
- String.valueOf(), 534-535
- String.watch(), 535-536
- stringToByte(), 1275, 1291-1292
- Style object, 650, 663, 689, 995
- Style.borderWidths(), 1004-1005
- Style.margins(), 1016-1017
- Style.paddings(), 1021
- Style.unwatch(), 1026
- Style.watch(), 1027
- sub() (String object), 499
- Submit object, 1030
- submit(), 761, 774, CD:1547, CD:1553
- Submit.blur(), 1031-1032
- Submit.click(), 1032-1033
- Submit.focus(), 1034
- Submit.handleEvent(), 1036
- Submit.unwatch(), 1042
- Submit.watch(), 1044
- substr() (String object), 499
- substring() (String object), 499
- substringData(), CD:1407-1409
- sup() (String object), 499
- System.out.println(), accessing, 121
- tags(), 639, 640-641
- taint(), 1045
- taintEnabled(), 21, 904, 912-913
- tan(), 409, 433
- Terminate(), CD:1972, CD:1978, CD:1985, CD:1989
- test(), 467, 488
- test(str), 89
- Text object, 1046, CD:1474
- Text.blur(), 1048
- Text.focus(), 1050
- Text.handleEvent(), 1052
- Text.select(), 1058
- Text.splitText(), CD:1474
- Text.unwatch(), 1060
- Text.watch(), 1062
- Textarea object, 1063
- Textarea.blur(), 1065
- Textarea.focus(), 1067
- Textarea.handleEvent(), 1070
- Textarea.select(), 1079
- Textarea.unwatch(), 1081
- Textarea.watch(), 1083
- TextStream object, CD:1912-1913
- TextStream.Close(), CD:1915
- TextStream.Read(), CD:1917
- TextStream.ReadAll(), CD:1918
- TextStream.ReadLine(), CD:1919
- TextStream.Skip(), CD:1919-1920
- TextStream.SkipLine(), CD:1920
- TextStream.Write(), CD:1921
- TextStream.WriteBlankLines(), CD:1922
- TextStream.WriteLine(), CD:1923
- toArray(), 55, CD:18291
- toExponential(), 440, 446
- toFixed(), 440, 447
- toGMTString(), 288, 321
- toLocaleLowerCase() (String object), 499
- toLocaleString(), 322, 440, 447-448
- toLocaleUpperCase() (String object), 500
- toLocalString() (Date object, 288
- toLowerCase() (String object), 500
- toPrecision(), 440, 448
- toSource()
 - Array object, 250, 267-268
 - Boolean object, 276-277
 - code, 362, 434, 449, 455-456, 488-489
 - Date object, 288, 322-323
 - Function object, 353
 - Math object, 409
 - Number object, 440
 - Object object, 453
 - String object, 500
- toString(), 1215-1216, 1252-1253, 1269
 - Array object, 42, 250, 268
 - arrays, code, 268-269
 - Boolean object, 274, 277-278
 - code, 363, 392, 406, 434-435, 449, 456, 489, 1269-1270
 - Connection object, 1200

- database object, 1234
- Date object, 288, 323-324
- DbPool object, 1257
- description, 268
- Function object, 353
- JSObject object, 397
- Math object, 409
- Number object, 440
- Object object, 453
- RegExp object, 467
- return value, 268
- String object, 500
- syntax, 268
- toString()
- toUpperCase() (String object), 500
- toUTCString() (Date object), 288, 324-325
- Transfer(), 1369, 1372
- ubound (VBAArray object), 551
- UIEvent object, CD:1761
- UIEvent.initUIEvent(), CD:1762
- unescape(), 366, 380, 548-549
- unlock(), 1301, 1305, 1312-1314, 1373, 1377
- unshift(), 42, 250, 269-270
- untaint(), 1084
- unwatch(), 633
 - Anchor object, 564
 - Area object, 573, 594-595
 - Array object, 250, 270
 - Boolean object, 274-278
 - Button object, 598
 - Checkbox object, 615
 - code, 450, 758-759, 805-806, 1270-1271, 1292-1294
 - Connection object, 1200
 - Cursor object, 1220
 - database object, 1234
 - Date object, 288, 325
 - DbPool object, 1257
 - Document object, 637
 - Event object, 698
 - File object, 1275
 - FileUpload object, 746
 - Form object, 761
 - Frame object, 779
 - Hidden object, 793
 - History object, 799
 - Image object, 808
 - Layer object, 822
 - Link object, 865
 - Location object, 886
 - Lock() object, 1301
 - Math object, 409
 - navigator object, 904
 - Number object, 440
 - Object object, 453
 - Option object, 915
 - Password object, 923
 - Plugin object, 934
 - project object, 1312
 - Radio object, 939
 - RegExp object, 467
 - Reset object, 953
 - Resultset object, 1345
 - Select object, 975
 - SendMail object, 1355
 - server object, 1373
 - Stproc object, 1388
 - String object, 500
 - Style object, 995
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1063
 - Window object, 1087
- updateRow(), 1220, 1231-1232
- URLEncode(), 1369, 1372
- UTC(), 288, 326-327
- valueOf()
 - Array object, 271
 - Boolean object, 274, 279
 - code, 271, 279, 327, 363-364, 458
 - Function object, 353
 - Number object, 440
 - Object object, 453
 - RegExp object, 467
 - syntax, 271, 279, 327
- VBAArray object, 551
- VBAArray.dimensions(), 552-553
- VBAArray.getItem(), 553-554
- VBAArray.lbound(), 555-556
- VBAArray.toArray(), 556-557
- VBAArray.ubound(), 557-558
- ViewCSS object, CD:1818
- ViewCSS.getComputedStyle(), CD:1818-1819
- watch(), 635-636
 - Anchor object, 564
 - Area object, 573, 596-597
 - Array object, 250, 271-272
 - Boolean object, 274, 279-280
 - Button object, 598

- Checkbox object, 615
- code, 451-452, 613-614, 716-717, 806-807, 1198-1199, 1218-1219
- Connection object, 1200
- Cursor object, 1220
- database object, 1234
- Date object, 288, 328
- DbPool object, 1257
- Document object, 637
- Event object, 698
- File object, 1275
- FileUpload object, 746
- Form object, 761
- Frame object, 779
- Hidden object, 793
- History object, 799
- Image object, 808
- Layer object, 822
- Location object, 886
- Math object, 409
- navigator object, 904
- Number object, 440
- Object object, 453
- Option object, 915
- Password object, 923
- Plugin object, 934
- project object, 1312
- Radio object, 939
- RegExp object, 467
- Reset object, 953
- ResultSet object, 1345
- Select object, 975
- SendMail object, 1355
- server object, 1373
- Stproc object, 1388
- String object, 500
- Style object, 995
- Submit object, 1030
- Text object, 1046
- Textarea object, 1063
- Window object, 1087
- Window object, 1085
- window.alert(), 1090
- window.atob(), 1091
- window.back(), 1092
- window.blur(), 1092
- window.btoa(), 1094
- window.captureEvents(), 1094-1096
- window.clearInterval(), 1096-1097
- window.clearTimeout(), 1098
- window.close(), 1099
- window.confirm(), 1102
- window.crypto.random(), 1104
- window.crypto.signText(), 1104-1105
- window.disableExternalCapture(), 1106
- window.enableExternalCapture(), 1110
- window.find(), 1111
- window.focus(), 1112
- window.forward(), 1113, 1171
- window.handleEvent(), 1115
- window.history property, 1117
- window.home(), 1118
- window.moveBy(), 1125
- window.moveTo(), 1126
- window.open(), 1134
- window.print(), 1146
- window.prompt(), 1147
- window.releaseEvents(), 1148-1149
- window.resizeBy(), 1150
- window.resizeTo(), 1151
- window.routeEvent(), 1152-1155
- window.scroll(), 1156
- window.scrollBy(), 1158-1159
- window.scrollTo(), 1160
- window.setHotKeys(), 1162
- window.setInterval(), 1162-1163
- window.setResizable(), 1164
- window.setTimeout(), 1164
- window.setZOptions(), 1165
- window.stop(), 1168
- window.unwatch(), 1171
- window.watch(), 1172
- write(), 167
 - code, 1295-1296, 1335, 1344, CD:1536, CD:1921-1922
 - Document object, 637
 - File object, 1275
 - HTMLDocument object, CD:1526
 - text, code to display, 694-695
 - window.document property, 1108
- Write(), 177, 1335, CD:1912
- WriteBlankLines(), CD:1912, CD:1922
- writeByte(), 1275, 1296-1297
- WriteLine(), CD:1912 CD:1923
- writeln()
 - code, CD:1537
 - Document object, 637
 - File object, 1275
 - files, code to write strings and carriage returns, 1297-1298
 - HTMLDocument object, CD:1526

- text, code to display, 695-696
- window.document property, 1108
- WScript object, CD:1929
- WScript.ConnectObject(), CD:1931
- WScript.CreateObject(), CD:1932
- WScript.DisconnectObject(), CD:1933
- WScript.Echo(), CD:1933
- WScript.GetObject(), CD:1934-1935
- WScript.Quit(), CD:1937
- WScript.Sleep(), CD:1939
- WSH (Windows Script Host), executing scripts, 184-185
- WshArguments object, CD:1944
- WshArguments.Count(), CD:1946
- WshArguments.Item(), CD:1946
- WshArguments.ShowUsage(), CD:1949
- WshController.CreateScript(), CD:1953
- WshEnvironment object, CD:1954
- WshEnvironment.Count(), CD:1956
- WshEnvironment.Item(), CD:1957
- WshEnvironment.Remove(), CD:1958
- WshNamed object, CD:1960
- WshNamed.Count(), CD:1961
- WshNamed.Exists(), CD:1961-1962
- WshNetwork object, CD:1964
- WshNetwork.AddPrinterConnection(), CD:1964-1965
- WshNetwork.AddWindowsPrinterConnection(), CD:1965
- WshNetwork.EnumNetworkDrives(), CD:1967
- WshNetwork.EnumPrinterConnection(), CD:1967
- WshNetwork.MapNetworkDrive(), CD:1968
- WshNetwork.RemoveNetworkDrive(), CD:1969
- WshNetwork.RemovePrinterConnection(), CD:1970
- WshNetwork.SetDefaultPrinter(), CD:1970
- WshRemote object, CD:1972
- WshRemote.Execute(), CD:1975
- WshRemote.Terminate(), CD:1978
- WshScriptExec object, CD:1985
- WshScriptExec.Terminate(), CD:1989
- WshShell object, CD:1990
- WshShell.AppActivate(), CD:1991
- WshShell.CreateShortcut(), CD:1991
- WshShell.Exec(), CD:1993-1994
- WshShell.ExpandEnvironmentStrings(), CD:1994

- WshShell.LogEvent(), CD:1995
- WshShell.Popup(), CD:1996
- WshShell.RegDelete(), CD:1997
- WshShell.RegRead(), CD:1998
- WshShell.RegWrite(), CD:1998-1999
- WshShell.Run(), CD:1999-2000
- WshShell.SendKeys(), CD:2000-2001
- WshShortcut object, CD:2005
- WshShortcut.Save(), CD:2009
- WshSpecialFolders object, CD:2013
- WshSpecialFolders.Count(), CD:2013-2014
- WshUnnamed.Count(), CD:2017
- WshUrlShortcut object, CD:2020
- WshUrlShortcut.Save(), CD:2021

Microsoft

- applications, JScript support, CD:1823
- ASP environment, 158
- core object hierarchy, 11
- ECMAScript (ECMA-262), 7
- JScript, 6
- Scripting Engine, CD:1823
- Scripting Runtime Library, CD:1824
- Web site, 28, 118, 184, CD:1824, CD:1927
- WSH (Windows Script Host), 7
- Microsoft Script Debugger, MSSD (Microsoft Script Debugger), 102-103**
- Microsoft Windows Script Host, WSH (Windows Script Host), 7, 16, 27-28, 179-189, CD:1936-1937**
- milliseconds argument (UTC() method), 326**
- milliseconds1 argument (Date object), 286**
- milliseconds2 argument (Date object), 286**
- MIME (Multipart Internet Mail Extension), 12**
- MimeType object, 899-900**
- MimeType.description property, 900**
- MimeType.enabledPlugin property, 901**
- MimeType.suffixes property, 901**
- MimeType.type property, 902**
- MimeType.unwatch() method, 903**
- MimeType.watch() method, 903**
- mimeType property, 904, 909**
- MIN VALUE property, 439, 442**
- min(), 409, 424-425**
- minorErrorCode() method**
 - code, 1207-1208
 - Connection object, 1199
 - database connection errors, code to retrieve, 1244-1245, 1265-1266

- database object, 1234
- DbPool object, 1257
- minorErrorMessage() method, 1266-1267**
 - code, 1208
 - Connection object, 1199
 - database connection errors, code to retrieve, 1245
 - database object, 1234
 - DbPool object, 1257
- minutes argument, 286, 326**
- models, DOM (Document Object Model), 12, 146-151**
- modes, script, code to retrieve, CD:1936**
- MODIFICATION constant (MutationEvent object), CD:1756**
- modifiers property, 698, 706-707**
- Modify control option (Server Side JavaScript Application Manager), 166**
- modular programming, 17**
- modularizing dates and times, 26**
- modulus assignment operator (%=), 209-210**
- modulus operator (%), 47, 208-209**
- monitors (17-inch) global variables, 39**
- monitorSpecs() function, 39**
- month argument, 286, 326**
- Motorola**
 - PowerPC processor, code to alert when on, 197-198
 - 680-0 processor, code to alert when on, 197
- mouse, CD:1732, CD:1737-1738, CD:1746, CD:1749-1755**
- MouseDown event, 91**
- MOUSEDOWN event (Event object), 699**
- MouseDown events, code, 677-678**
- MOUSEDOWN property, code to access, 734**
- MouseEvent object, CD:1748, CD:1756**
- MouseEvent.altKey property, CD:1749**
- MouseEvent.button property, CD:1749**
- MouseEvent.clientX property, CD:1750**
- MouseEvent.clientY property, CD:1751**
- MouseEvent.ctrlKey property, CD:1751**
- MouseEvent.initMouseEvent() method, CD:1752**
- MouseEvent.metaKey property, CD:1753**
- MouseEvent.relatedTarget property, CD:1753**
- MouseEvent.screenX property, CD:1754**
- MouseEvent.screenY property, CD:1754-1755**
- MouseEvent.shiftKey property, CD:1755**
- MouseMove event, 91**
- MOUSEMOVE**
 - event (Event object), 699
 - property, code to access, 735
- MouseOut event, 91**
- MOUSEOUT**
 - event (Event object), 699
 - property, code, 736
- MouseOver event, 91**
- MOUSEOVER**
 - event (Event object), 699
 - property, code to access, 736-737
- MouseUp event, 91, 678-679**
- MOUSEUP**
 - event (Event object), 699
 - property, code to access, 737-738
- Move event, 91**
- MOVE**
 - event (Event object), 699
 - property, code to access, 738-739
- Move() method, CD:1845, CD:1853, CD:1891, CD:1900-1901**
- moveAbove() method, 822, 839**
- moveBelow() method, 822, 840**
- moveBy() method, 822, 840-841, 1086, 1125-1126**
- MoveFile() method (FileSystemObject object), CD:1865, CD:1887-1888**
- moveFirst() method, 337-340**
- MoveFolder() method (FileSystemObject object), CD:1865, CD:1888-1889**
- moveNext() method, 337, 340-341**
- moveTo() method, 822, 841-842, 1086, 1127**
- moveToAbsolute() method, 822, 842-843**
- moving**
 - enumerators, code, 339-340
 - files to new directory, code, CD:1853
 - to drives, code, 340-341
- MSSD (Microsoft Script Debugger), 102-103**
- multidimensional arrays, 39, 42-43**
- multiline comments (/*/), 216**
- multiline property, 467, 484-485**
- Multipart Internet Mail Extension (MIME), 12**
- multiple property, CD:1659, CD:1664**

multiplication assignment operator (*=),
213-214
multiplication operator (*), 47, 213
multiplication tables, 66-67
mutation events, code, CD:1757, CD:1759
MutationEvent object, CD:1756
MutationEvent.attrChange property,
CD:1758
MutationEvent.attrName property,
CD:1758
MutationEvent.initMutationEvent()
method, CD:1759
MutationEvent.newValue property,
CD:1760
MutationEvent.prevValue property,
CD:1760
MutationEvent.relatedNode property,
CD:1757
myButton event, 93

N

\n escape sequence, 36
name attribute, HTML tags, 147
name parameter, 171, CD:1869
name property
Anchor object, 564
Attr object, CD:1404
attributes, code to write names, CD:1404
Button object, 597
button 's name, code to access, 605-606
check box names, code to display,
626-627
Checkbox object, 615
code to access, 770-771
code, 795, 814, 927-928, 1128, CD:1427
DocumentType object, CD:1425
Error object, 342
FileUpload object, 745
Form object, 761
Frame object, 778
Hidden object, 793
HTMLAnchorElement object, CD:1480
HTMLAppletElement object, CD:1490
HTMLButtonElement object, CD:1513
HTMLFormElement object, CD:1547
HTMLFrameElement object, CD:1554
HTMLIFrameElement object, CD:1570
HTMLImageElement object, CD:1577
HTMLInputElement object, CD:1586
HTMLMapElement object, CD:1615
HTMLMetaElement object, CD:1618

HTMLObjectElement object, CD:1624
HTMLParamElement object, CD:1649
HTMLSelectElement object, CD:1659
HTMLTextAreaElement object,
CD:1715
Image object, 807
Layer object, 822
Password object, 922
Plugin object, 934
Radio object, 938, 946
Reset object, 952
select box names, code to retrieve,
983-984
Select object, 975
Submit button names, code to retrieve,
1037-1038
Submit object, 1030
text areas, names, code to retrieve,
1071-1072
text boxes, names, code to retrieve, 1054
Text object, 1046
Textarea object, 1064
Window, 1087

Name property

File object, CD:1845, CD:1854
Folder object, CD:1891
WScript object, CD:1929
WSH name, code to retrieve, CD:1936

Named property, CD:1944, CD:1948-1949
namedItem() method, CD:1518, CD:1521

NamedNodeMap.getNamedItem()
method, CD:1448

NamedNodeMap.getNamedItemNS()
method, CD:1449

NamedNodeMap.item() method,
CD:1449-1450

NamedNodeMap.length property, CD:1450

NamedNodeMap.removeNamedItem()
method, CD:1451

NamedNodeMap.removeNamedItemNS()
method, CD:1451

NamedNodeMap.setNamedItem() method,
CD:1452

NamedNodeMap.setNamedItemNS()
method, CD:1453

names, 344, 605-606, 626-627, 657-658,
1223-1224

NAMESPACE_ERR constant
(DOMException object), CD:1430
namespaceURI property, CD:1455,
CD:1463

- naming, variables, 37
- NaN (Not A Number), 45
 - object, 376, 436
 - property, code, 365, 439, 443
- native keyword, 436
- Nav (Netscape Communicator), 563
- navigator object, 12, 904-905
- navigator.appCodeName property, 905
- navigator.appName property, 906
- navigator.appVersion property, 906
- navigator.javaEnabled() method, 907
- navigator.language property, 908
- navigator.mimeTypes property, 908-909
- navigator.platform property, 909
- navigator.plugins property, 910
- navigator.plugins.refresh() method, 911
- navigator.preference() method, 911
- navigator.taintEnabled() method, 912
- navigator.unwatch() method, 914
- navigator.userAgent property, 913
- navigator.watch() method, 914
- NEGATIVE INFINITY property, 440, 444
- NES (Netscape Enterprise Server), 563
- nested if...else statements, code for readability, 64
- nested loops, continue statements, 75
- Netscape, 7, 11, 22, 159-167, 437, 460-463
 - Communicator (Nav), 563
 - DevEdge Web site, 21-22
 - Enterprise Server (NES), 563
 - Navigator, 20-23, 108-111
 - Navigator 2.02, 109
 - Navigator 4, continue keyword error, 73
 - Navigator 6, 110
- networks, CD:1967-1968
- new keyword, 86
- new operator, 437-438
- newsgroups, 30
- newValue property, CD:1756, CD:1760-1761
- next() method, 1220, 1227-1228, 1345, 1349-1350
- next property, 799, 803-804, 1117
- nextSibling property, CD:1455, CD:1464
- NO DATA ALLOWED ERR constant (DOMException object), CD:1429
- NO MODIFICATION ALLOWED ERR constant (DOMException object), CD:1429
- Node object, CD:1453-1456
- Node.appendChild() method, CD:1457
- Node.attributes property, CD:1456
- Node.childNodes property, CD:1457
- Node.cloneNode() method, CD:1458
- Node.firstChild property, CD:1459
- Node.hasAttributes() method, CD:1459
- Node.insertBefore() method, CD:1460
- Node.isSupported() method, CD:1461
- Node.lastChild property, CD:1462
- Node.localName property, CD:1462
- Node.namespaceURI property, CD:1463
- Node.nextSibling property, CD:1463
- Node.nodeName property, CD:1464
- Node.nodeType property, CD:1465
- Node.nodeValue property, CD:1465
- Node.normalize() method, CD:1466
- Node.ownerDocument property, CD:1466-1467
- Node.parentNode property, CD:1467
- Node.prefix property, CD:1468
- Node.previousSibling property, CD:1468
- Node.removeChild() method, CD:1469
- Node.replaceChild() method, CD:1469-1470
- nodeName property, CD:1455, code, CD:1464
- nodes, 149, CD:1424
- nodeType property, CD:1455, CD:1465
- nodeValue property, CD:1455, CD:1465-1466
- noHref property CD:1500-1501, CD:1497
- non-identity operator (!==), 58, 207-208
- noResize property, code, CD:1554, CD:1559
- normalize() method, code, CD:1456, CD:1466
- noShade property, code, CD:1565, CD:1566-1567
- NOT (!) logical operator, 55
- NOT (-) bitwise operator, 60
- NOT (~) bitwise operator, 224
- Not a Number. *See* NaN
- not equal operator, 57, 206-207
- NOT FOUND ERR constant (DOMException object), CD:1429
- NOT SUPPORTED ERR constant (DOMException object), CD:1430
- NOTATION NODE constant (Node object), CD:1454
- Notation object, CD:1470
- notation, dot, 93
- Notation.publicId property, CD:1471

Notation.systemId property, CD:1471
notationName property, CD:1445-1446
notations
 exponential, 34
 property, reading, code, CD:1425,
 CD:1428
Notepad, keystrokes, code to send,
 CD:2002
noWrap property, code, CD:1674,
 CD:1681
null keyword, 36, 438
num argument (Button object), 597
num parameter (GetSpecialFolder()
 method), CD:1886
num1 argument (Checkbox object), 615
num2 argument (Checkbox object), 615
number argument (Error object), 341
Number constructor, code, 440
Number() method, 377, 438-439
Number object, 35, 439-440
number of arguments, retrieving, code,
 CD:1961
number of drives, code to access, CD:1843
number of plug-ins, accessing, 119
NUMBER OF PROCESSORS (Windows
 environment variable), CD:1955
Number property, 365, 1183, 1187,
 CD:1979, CD:1982-1983
number property (Error object), 342
Number.constructor property, 440-441
Number.MAX_VALUE, 35, 441
Number.MIN_VALUE, 35, 442
Number.NaN, 35, 443
Number.NEGATIVE_INFINITY, 35, 443
Number.POSITIVE_INFINITY, 35, 444
Number.prototype property, 445
Number.toExponential() method, 446
Number.toFixed() method, 447
Number.toLocaleString() method, 447
Number.toPrecision() method, 448
Number.toSource() method, 448
Number.toString() method, 449
Number.unwatch() method, 449-550
Number.valueOf() method, 450-451
Number.watch() method, 451
number1 variable, values, assigning, 54
number2 variable, values, assigning, 54
numbers, 34-35, 45, 50, 195, 225, 345, 665,
 1104

O

O subfield (ClientCertificate collection),
 1320
-o outfile command-line parameter, 161
Object Central Web site, 10
Object object, 452-453
Object() constructor, arrays, creating,
 44-45
object-oriented programming (OOP), 7-17
Object.constructor property, 453
Object.eval() method, 454
Object.prototype property, 455
Object.toSource() method, 455
Object.toString() method, 456
Object.unwatch() method, 457
Object.valueOf() method, 458
Object.watch() method, 458
ObjectContext object, 168, 1308-1309
ObjectContext.OnTransactionAbort event,
 1309
ObjectContext.OnTransactionCommit
 event, 1310
ObjectContext.SetAbort() method,
 1310-1311
ObjectContext.SetComplete() method,
 1311
objects, 934, CD:1532
 accessing, 16-17
 ActiveXObject, 241-242
 Anchor, 564-565
 Applet, 572
 Application, 168, 1177-1178
 Area, 573-597
 Arguments, 243-249
 Array, 42, 249-251, 263-266, 269
 arrays as, 43-44
 ASPErrors, 168, 1183
 Attr, CD:1403-1404
 Automation, 272
 based on other objects, creating, 9
 based programming, JavaScript, 10-16
 BaseElement, code to create,
 CD:1539-1540
 Big Button, properties, 71
 BLOB (binary large objects), 1188
 blob, 1189
 Boolean, 273-274
 Button, 71, 597-598
 Car, 298, 402-405
 CDATASection, CD:1406-1407

- CharacterData, CD:1407
- Checkbox, 614-615
- children, 8
- client, 1193-1197
- client-side JavaScript object
 - hierarchy, 11
- COM (Component Object Model), CD:1824
- Comment, CD:1412
- Connection, 1199-1201
- core, 10-11
- Counter, CD:1767-1768
- createEvent() method, code, CD:1732
- creating, 16
- CSSCharsetRule, CD:1770
- CSSFontFaceRule, CD:1771
- CSSImportRule, CD:1772
- CSSMediaRule, CD:1774-1775
- CSSPageRule, CD:1778
- CSSPrimitiveValue, CD:1779-1781
- CSSRule, CD:1786-1787
- CSSRuleList, CD:1790, CD:1794, CD:1806, CD:1817
- CSSStyleDeclaration, CD:1792-1793
- CSSStyleRule, CD:1799
- CSSStyleSheet, CD:1800-1801
- CSSUnknownRule, CD:1804
- CSSValue, CD:1804-1807
- Cursor, 1219-1221
- database, 13, 1233-1235
- Date, 285-288, 323
- DbPool, 13, 170, 1255-1258, 1262-1263
- Dictionary, 1825-1833
- Document, 636-637, CD:1412-1413
- document, 19-21
- DocumentCSS, CD:description or syntax, CD:1809
- DocumentCSSList, getOverrideStyle() method or methods, CD:1809
- DocumentEvent, CD:1731-1732
- DocumentEvent.createEvent() method, CD:1732
- DocumentFragment, CD:1424-1415
- DocumentType, CD:1425
- DOM (Document Object Model), 12, CD:1429-1431, CD:1810
- Drive (JScript3.0+), CD:1832-1842
- Element, CD:1434-1435
- ElementCSSInlineStyle, CD:1811
- Embed, 697
- embedded, 654-655, 697
- Entity, CD:1445
- EntityReference, CD:1447
- Enumerator, 336-337
- Error, 341-345
- Event, 698-699, 707, CD:1732-1733, CD:1741-1744, CD:1748, CD:1756
- File, 174-176, 1275-1277, CD:1845-1860
- Files, CD:1861-1864
- FileSystemObject, CD:1833-1834, CD:1864-1890
- FileUpload, 745-747
- Folder, CD:1891-1892
- folder, code to retrieve, CD:1884
- for...in, 351
- Form, 658-763
- Frame, 778-779
- FSO (File System Objects), CD:1833
- Function, 353
- functions as, 80
- Global, 365-366, 376
- Hidden, 793-794
- History, 799
- history, data tainted by default, 21
- HTMLAnchorElement, CD:1480-1489
- HTMLAppletElement, CD:1489-1497
- HTMLAreaElement, CD:1497-1503
- HTMLBaseElement, CD:1503-1505
- HTMLBaseFontElement, CD:1505-1507
- HTMLBodyElement, CD:1507-1512
- HTMLBRElement, CD:1512-1513
- HTMLButtonElement, CD:1513-1518
- HTMLCollection, CD:1518-1521
- HTMLDirectoryElement, CD:1521-1522
- HTMLDivElement, CD:1522-1523
- HTMLDListElement, CD:1524-1525
- HTMLDocument, CD:1525-1526
- HTMLDOMImplementation, CD:1537-1538
- HTMLElement, CD:1539-1540
- HTMLFieldSetElement, CD:1543-1544
- HTMLFontElement, CD:1544-1546
- HTMLFormElement, CD:1546-1554
- HTMLFrameElement, CD:1554-1560
- HTMLFrameSetElement, CD:1560-1562
- HTMLHeadElement, CD:1562-1563
- HTMLHeadingElement, CD:1564-1565
- HTMLHRElement, CD:1565-1568
- HTMLHtmlElement, CD:1568-1569

HTMLIFrameElement, CD:1569-1577
HTMLImageElement, CD:1577-1585
HTMLInputElement, CD:1585-1600
HTMLIsIndexElement, CD:1600-1602
HTMLLabelElement, CD:1602-1604
HTMLLegendElement, CD:1604-1607
HTMLLIElement, CD:1607-1609
HTMLLinkElement, CD:1609-1615
HTMLMapElement, CD:1615-1617
HTMLMenuElement, CD:1617-1618
HTMLMetaElement, CD:1618-1621
HTMLModElement, CD:1622-1623
HTMLObjectElement, CD:1624-1636
HTMLLOListElement, CD:1636-1638
HTMLOptGroupElement,
 CD:1639-1641
HTMLOptionElement, CD:1641-1647
HTMLParagraphElement,
 CD:1647-1649
HTMLParamElement, CD:1649-1652
HTMLPreElement, CD:1652-1653
HTMLQuoteElement, CD:1653-1654
HTMLScriptElement, CD:1654-1659
HTMLSelectElement, CD:1659-1670
HTMLStyleElement, CD:1670-1673
HTMLTableCaptionElement,
 CD:1673-1674
HTMLTableCellElement, CD:1674-1683
HTMLTableColElement, CD:1683-1688
HTMLTableElement, CD:1688-1702
HTMLTableRowElement,
 CD:1702-1709
HTMLTableSectionElement,
 CD:1709-1714
HTMLTextAreaElement, CD:1714-1714
HTMLTitleElement, CD:1724-1725
HTMLULListElement, CD:1725-1727
Image, 807-809
image, 19-21
java, 389-395
JScript RunTime object hierarchy, 15
JSException, 395
JSObject, 396-397
KEYPRESS, code to access, 730-731
lastName, 86
Layer, 667-668, 821-823
layer, origin verification, 19
Link, 21, 864-866
Location, 885-886
location, 19-21
Lock(), 1300-1301
Math, 34-35, 46, 408-409
MimeType, 899-900
models, 181
MouseEvent, CD:1748, CD:1756
MutationEvent, CD:1756
NaN, 376, 436
navigator, 12, 904-905
netscape, 437
Node, CD:1453-1456
Notation, CD:1470
Number, 35, 439-440
Object, 168, 452-453, 1308-1309
Option, 21, 914-915
orientation of JavaScript, 10-16
Packages, 121, 460-461
parent, 8
passing origin verification, 19
Password, 922-923
Plugin, data tainted by default, 21
ProcessingInstruction, CD:1472
project, 1312-1313
property, CD:1490, CD:1495-1496
prototype property, 16
Radio, 938-939
Rect, CD:1812-1813
RegExp, 86-89, 466-468
Request, 168, 176, 1318, 1327-1328
Reset, 952-953
Response, 168, 177, 1334-1335
Resultset, 1344-1346
RGBColor, CD:1815-1816
screen, 967-968
Scripting.Signer, CD:1927-1928
ScriptingContext, 1354
Select, 974-975
SendMail, 172-173, 1354-1357
Server, 13, 168, 1368-1369, 1373
Session, 168, 1378-1379
Stproc, 1388-1389
String, 497-501
Style, 649-650, 662-663, 688-389,
 994-997
Submit, 1029-1030, 1043-1044
sun, 536-537
tainted by default, 20
Text, 1046-1047, CD:1474
Textarea, 1063-1065
TextStream, CD:1912-1913
toSring(), length method, 390
UIEvent, CD:1761
unnamed, code to display,
 CD:2016-2017

- VBArray, 551-552, CD:1829
- vehicle, 8-9
- ViewCSS, CD:1818
- Window, 12, 1085-1090, 1134-1136
- window, 19-21, 139
- WScript, 182-183, CD:1929-1930, CD:1943-1944
- WSH (Windows Script Host), 16, 182-184
- WshArguments, CD:1944-1945, CD:1951-1952
- WshController, CD:1952-1954, CD:1973
- WshEnvironment, CD:1954-1956
- WshNamed, CD:1960
- WshNetwork, 183-184, CD:1933, CD:1963-1964
- WshRemote, CD:1972-1973, CD:1976, CD:1979
- WshScriptExec, CD:1985
- WshShell, 183, CD:1990-1991, CD:2003-2004
- WshShortcut, CD:2004-2005
- WshSpecialFolders, CD:2012-2013
- WshUnnamed, CD:2015-2017
- WshUrlShortcut, CD:2019-2020
- WshURLShortcut, CD:2020
- octal integers, 34**
- offscreenBuffering property, 1087, 1129**
- onAbort event handler, 92, 808, 814-815**
- onBlur event handler, 92, 606, 627-628, 784-785, 844-845**
 - Button object, 598
 - Checkbox object, 615
 - Password object, 923
 - Radio object, 939
 - Reset object, 953
 - Select object, 974
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1063
- onBlur property, 745, 778, 882**
- onChange event handler, 92, 754-755, 974, 985-986**
- onChange property (FileUpload object), 745**
- onClick event handler, 92**
 - alert boxes, code to display, 607
 - Button object, 598
 - Checkbox object, 615, 630
 - click events, code, 672
 - code, 871, 947-948, 960
 - Document object, 637
 - messages, code to display, 629-630
 - Radio object, 939
 - Reset object, 953
 - Submit object, 1030
 - textboxes, contents, code to display, 1039-1040
 - window.document property, 1108
- onclick property, 93**
- onClick property, 865**
- onDbClick event handler, 92, 673, 872**
 - Area object, code, 574, 585
 - Document object, 637
 - window.document property, 1108
- onDbClick property (Link object), 865**
- onDragDrop event handler, code, 92, 1130**
- one-dimensional arrays, creating, 40**
- onError event handler, 92, 808, 815, 1131**
- onFocus event handler, 92, 607-608, 630-631, 755-756, 785**
 - Button object, 598
 - Checkbox object, 615
 - Password object, 923
 - Radio object, 939
 - Reset object, 953
 - Select object, 975
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1063
- onFocus property**
 - FileUpload object, 745
 - Frame object, 778
 - Layer object, 822
- onKeyDown event handler, 92, 674, 815, 872, 1076**
 - Document object, 637
 - oImage object, 808
 - Textarea object, 1063
 - window.document property, 1108
- onKeyDown property (Link object), 865**
- onKeyPress event handler, 92, 675, 816, 873, 1076-1077**
 - Document object, 637
 - Image object, 808
 - Textarea object, 1063
 - window.document property, 1108
- onKeyPress property (Link object), 865**
- onKeyUp event handler, 92, 816, 874, 1077-1078**
 - Document object, 637
 - Image object, 808

- KeyUp events, code, 676
- Link object, 865
- onTextarea object, 1063
- window.document property, 1108
- onLoad event handler, 92, 118, 677, 817, 846-847, 1132**
 - Document object, 637
 - Image object, 808
- onLoad property (Layer object), 822**
- onMouseDown event handler, 92, 677-678, 874-875**
 - Button object, 598
 - Document object, 637
 - window.document property, 1108
- onMouseDown property (Link object), 865**
- onMouseMove event handler, 92**
- onMouseOut**
 - event handler, 92, 574, 585-586, 847-848, 875-876
 - property, 822, 865
- onMouseOver**
 - event handler, 92, 574, 586-587, 848-849, 876-877
 - property, 822, 865
- onMouseUp**
 - event handler, 92, 598, 637, 678-679, 877, 1108
 - property (Link object), 865
- onMove**
 - event handler, 92, 786, 1132
 - property (Frame object), 778
- onOnMouseDown event handler, alert boxes, code to display, 608**
- onOnMouseUp event handler, alert boxes, code to display, 609**
- onReset**
 - event handler, code, 92, 771-772
 - property (Form object), 761
- onResize**
 - event handler, code, 92, 786-787, 1133
 - property (Frame object), 778
- onSelect event handler, 92, 1046, 1058, 1063, 1078-1079**
- onSubmit**
 - event handler, 92
 - property (Form object), 761
- onSubmit() method, code, 772-773**
- OnTransactionAbort event, 1308-1310**
- OnTransactionCommit event, 1308-1310**
- onUnload event handler, 92, 637, 680, 1133**
- OOP (object-oriented programming), 7-17**

- open() method, 141, 175, 637, 680, 1108**
- OpenAsTextStream() method (File object), CD:1845, CD:1854-1856**
- opener property, 1087, 1138-1139**
- opening**
 - documents, code, 680
 - files, code, 1286
- OpenTextFile() method, CD:1865, CD:1889-1890**
- Opera, Web site, 108,-109 114, 563**
- operations, OR (|), returning values, 60**
- operators**
 - % (modulus), 47
 - %= (assignment), 53
 - &= (assignment), 53
 - * (multiplication), 47
 - *= (assignment), 52
 - + (addition), 46, 49-50, 53
 - ++ (post-increment), 48
 - ++ (pre-increment), 48
 - += (assignment), 52
 - (subtraction), 46-47
 - (unary), 49
 - = (assignment), 52
 - (post-decrement), 49
 - (pre-decrement), 48-49
 - / (division), 47, 215
 - /= (assignment), 52
 - = (assignment), 37
 - addition (+), 225
 - addition assignment (+=), 227-228
 - arithmetic, 45-50
 - assignment, 50-54
 - assignment (=), 232
 - bitwise, 59-61
 - AND (&), 210-211
 - AND assignment (&=), 212-213
 - exclusive OR (^), 219
 - exclusive OR assignment (^=), 220-221
 - NOT (~), 224
 - OR (|), 221
 - OR assignment (|=), 223-224
 - comparison, 55-58
 - conditional (?), 58, 218-219
 - decrement (--), 204-205
 - delete, 331-332
 - division assignment (/=), 217
 - equal (==), 233-234
 - greater than (>), 235-236
 - greater than or equal (>=), 236-237

identity (===), 234-235
 increment (++), 226-227
 less than (<), 228-229
 less than or equal (<=), 231-232
 logical, 54-55
 AND (&&), 211-212
 NOT (!), 205-206
 OR (||), 222-223
 Math object, methods, 46
 modulus (%), 208-209
 modulus assignment (%=), 209-210
 multiplication assignment (*=), 213-214
 new, 437-438
 non-identity (!==), 207-208
 not equal (!=), 206-207
 OR (|), 646
 precedence, 61-62
 shift left (<<), 229-230
 shift left assignment (<<=), 230-231
 shift right with sign (>>), 237-238
 shift right with sign assignment (>>=), 238-239
 shift right zero fill (>>>), 239-240
 shift right zero fill assignment (>>>=), 240-241
 string, + (addition operator), 49-50
 subtraction (-), 202-203
 subtraction assignment (-=), 232-233
 unary, 546-547
 unary negation, 203
 unary negation (-), 203-204
 void, code or description, 559
 [] (square brackets), 40
 <<= (assignment), 53
 >>= (assignment), 53
 >>>= (assignment), 53
 ^= (assignment), 53
 |= (assignment), 53

Option object, 21, 914-915

/option (jsac), 160

-option (jsac), 160

Option.defaultSelected property, 916

Option.selected property, 917

Option.text property, 918

Option.unwatch() method, 919

Option.value property, 920

Option.watch() method, 921

optionb option (open() method), 1276, 1286

options, 975, 988-989, 1286, CD:1659, CD:1665-1666

OR (|) logical operator, 55

OR (|) bitwise operator, 60

OR (|) operation, returning values, 60

OR (|) operator, 646

Organization property, 1355, 1362

origin verification, objects passing, 19

OS (Windows environment variable), CD:1955

OS/2, 20, 109

OU subfield (ClientCertificate collection), 1320

outerHeight

parameter (Window object), 1135

property, code, 1139-1140

property (Window), 1087

outerWidth

parameter (Window object), 1135

property, code, 1141

property (Window), 1087

outParamCount() method, 1388, 1390-1391

outParameters() method, 1388, 1391-1392

output, 127, 651-652

overriding, style, code, CD:1809

overwrite parameter, CD:1867-1870

ownerDocument property, CD:1455, CD:1467

ownerElement property, CD:1404-1406

ownerRule property, CD:1801-1803

P

-p path command-line parameter, 161

package keyword, 459

packages, 462-434

Packages object, 121, 460-461

Packages.className property, 461

Packages.java

property, 461-462

subpackage, code, 462

Packages.netscape property, 462

Packages.sun property, 463

paddingBottom property, 649, 663, 688, 996, 1019

paddingLeft property, code, 649, 663, 688, 996, 1019-1020

paddingRight property, code, 649, 663, 688, 996, 1020-1021

padding() method, 650, 663, 689, 995, 1021-1022

paddingTop property, 649, 663, 688, 996, 1022-1023

Page Signer tool, downloading, 22**pages**

- ASPs (Active Server Pages), 24-25, 168-169 177
- buffers, code to flush to, 1298
- clicks, code, 1052-1053, 1070-1071
- defaultStatus property, code, 1106
- e-mail, building, 174
- print() method, code, 1146-1147
- status property, code, 1166-1169
- visited, code to access, 1114-1118

pageX property, 698, 707-708, 822, 849-850**pageXOffset property, code, 1088, 1142-1143****pageY property, 698, 709, 822, 850-851****pageYOffset property, 1088, 1143-1144****parameters**

- align (blobImage() method), 1190
- altText (blobImage() method), 1190
- alwaysLowered (Window object), 1134
- alwaysRaised (Window object), 1134
- Arguments property, command-line, CD:1931
- blobImage() method, 1190
- border (blobImage() method), 1190
- color, code to display, CD:1948-1949
- command-line, jsac (JavaScript Application Compiler), 160-161
- CommitFlag, 171
- commitFlag
 - connect() method, 1238, 1259
 - DbPool object, 1256
 - DbPool() method, 1262
- connect() method, 1238, 1258
- create (OpenTextFile() method), CD:1890
- DBInstance, 171
- dbInstance
 - connect() method, 1238, 1258
 - DbPool object, 1256
 - DbPool() method, 1262
- DBName, 171
- dbName
 - connect() method, 1238, 1259
 - DbPool object, 1256
 - DbPool() method, 1262
- DbPool object, 1256
- DbPool() method, 1262
- DBType, 171

dbType

- connect() method, 1238, 1258
 - DbPool object, 1256
 - DbPool() method, 1262
- dependent (Window object), 1134
- destination
 - CopyFile() method, CD:1867
 - CopyFolder() method, CD:1868
- directories (Window object), 1134
- drive
 - GetDrive() method, CD:1879
 - GetDriveName() method, CD:1880
- filename
 - CreateTextFile() method, CD:1870
 - DeleteFile() method, CD:1871
 - FileExists() method, CD:1875
 - GetFileName() method, CD:1883
 - OpenTextFile() method, CD:1890
- fileType (blobImage() method), 1190
- foldername
 - DeleteFolder() method, CD:1872
 - FolderExists() method, CD:1876
 - GetParentFolderName() method, CD:1885
- force
 - DeleteFile() method, CD:1871
 - DeleteFolder() method, CD:1872
- format, CD:1855, CD:1890
- height
 - blobImage() method, 1190
 - Window object, 1135
- hotkeys (Window object), 1135
- innerHeight (Window object), 1135
- innerWidth (Window object), 1135
- iomode, CD:1855, CD:1890
- ismap (blobImage() method), 1190
- letter (DriveExists() method), CD:1873
- location (Window object), 1135
- MaxConn, 171
- maxConn
 - connect() method, 1238, 1259
 - DbPool() method, 1262
 - DbPool object, 1256
- menubar (Window object), 1135
- name, 171, CD:1869
- num (GetSpecialFolder() method), CD:1886
- outerHeight (Window object), 1135
- outerWidth (Window object), 1135

- overwrite
 - CopyFile() method, CD:1867
 - CopyFolder() method, CD:1868
 - CreateTextFile() method, CD:1870
- path (GetExtensionName() method), CD:1881
- PWD, 171
- pwd
 - connect() method, 1238, 1258
 - DbPool() method, 1262
 - DbPool object, 1256
- resizable (Window object), 1135
- screenX (Window object), 1135
- screenY (Window object), 1135
- scrollbars (Window object), 1135
- source
 - CopyFile() method, CD:1867
 - CopyFolder() method, CD:1868
- timeout, 171
- titlebar (Window object), 1135
- toolbar (Window object), 1135
- UID, 171
- uid
 - connect() method, 1238, 1258
 - DbPool() method, 1262
 - DbPool object, 1256
- unicode (CreateTextFile() method), CD:1870
- unnamed, code to display, CD:1951, CD:2017-2018
- width
 - blobImage() method, 1190
 - Window object, 1135
- Window object, 1134
- WScript, code to display, CD:1946-1947
- z-lock (Window object), 1136
- parent/child relationship, windows, 139-141**
- ParentFolder property, CD:1845, CD:1856, CD:1891, CD:1902**
- parentheses (), 61**
- parentLayer property, 822, 851-852**
- parentNode property, CD:1455, CD:1467**
- parentRule property, CD:1787-1788, CD:1792, CD:1797**
- parents**
 - check boxes, 624
 - code, windows, 142, 144
 - folders, names, code to obtain, CD:1885
 - objects, 8
 - of buttons, code to access, 602-603
 - property, 778, 1088, 1144
- parentStyleSheet property, CD:1787-1789**
- parse() method (Date object), 287, 306**
- parseFloat() method, 366, 377-378, 464, 1193**
- parseInt() method, 366, 378-379, 465, 1193**
- passing, 577-578, 625**
 - arguments to displayArguments() function, code with arguments array, 79
 - events, code, 660-661, 685-686
- Password object, 922-923**
- Password.blur() method, 923**
- Password.defaultValue property, 924**
- Password.focus() method, 925**
- Password.form property, 925-926**
- Password.handleEvent() method, 926**
- Password.name property, 927**
- Password.onBlur event handler, 928**
- Password.onFocus event handler, 929**
- Password.select() method, 930**
- Password.type property, 930**
- Password.unwatch() method, 931**
- Password.value property, 932**
- Password.watch() method, 933**
- passwords, validating, code, 98-101**
- PATH (Windows environment variable), CD:1955**
- PATH INFO variable, 1325, 1386**
- Path property, CD:1833, CD:1838-1839, CD:1845, CD:1857, CD:1891**
- PATH TRANSLATED variable, 1325, 1387**
- PATHEXT (Windows environment variable), CD:1955**
- pathname property, 573, 587-588, 865, 878, 886, 891-892**
- paths, CD:1858-1859, CD:1955-1957**
 - attribute, cookies, 137
 - parameter (GetExtensionName() method), CD:1881
- patterns, matching, 85-89**
- people's favorite foods, displaying, 243-244**
- Personal Web Server (PWS), 1175**
- personalbar property, 1088, 1145-1146**
- PI property, 408, 425-426**
- PICS property, code, 1343**
- pipe (|) (pattern matching character), 87**
- pixelDepth property, accessing, code, 972**
- placeholders, labels, 74**
- platform**
 - independence, 24
 - property, 904, 909-910

- plug-ins, 118-120, 681-682
- Plugin object, 21, 934
- Plugin.description property, 935
- Plugin.filename property, 935
- Plugin.length property, 936
- Plugin.name property, 937
- Plugin.unwatch() method, 937
- Plugin.watch() method, 938
- plugins property, 636, 681, 904, 910, 1109
- plugins.length property, plug-ins, code to display, 682
- plugins.refresh() method, 904, 911
- plus sign (+), 87, 563
- pools, 170-171, 1210-1211
- pop() method, 42, 250, 257
- pop-up
 - display boxes, WSH (Windows Script Host), 186
 - message boxes, button types or icon types, CD:1996
 - windows, for games, 139
- Popup() method, CD:1990, CD:1996-1997
- port property, code, 573, 588-589, 865, 879, 886
- positions, 253-256
- POSITIVE INFINITY property, 440, 444-445
- POST method, 124-125
- post-decrement, 48-49, 205, 227
- pow() method, code, 409, 426-427
- pre-decrement, comparing, 48-49, 205, 227
- precedence (operators), 61-62
- preference() method, 904, 912
- Preferences command (Edit menu), 136
- prefix property, CD:1455, CD:1468
- preventDefault() method, CD:1733, CD:1738
- previous property, 799, 804-805, 1117
- previousSibling property, CD:1455, CD:1468-1469
- prevValue property, CD:1756, CD:1760-1761
- prices, displaying, code, 666-667
- primitive Boolean value, versus Boolean object, code, 274
- primitiveType property, CD:1781, CD:1785
- print() method, 779, 788, 1086, 1146-1147
- printers, 184, CD:1968
- private keyword, 465-466
- privileges, 22-23

- problem code, debugging, 104
- procedures, stored, 1214-1215, 1250-1251, 1268
- processing forms, 124-136
- PROCESSING INSTRUCTION NODE
 - constant (Node object), CD:1454
- ProcessingInstruction object, CD:1472
- ProcessingInstruction.data property, CD:1472-1473
- ProcessingInstruction.target property, CD:1473
- PROCESSOR ARCHITECTURE (Windows environment variable), CD:1955
- PROCESSOR IDENTIFIER (Windows environment variable), CD:1955
- PROCESSOR LEVEL (Windows environment variable), CD:1955
- PROCESSOR REVISION (Windows environment variable), CD:1955
- processors, 194, 197-201
- profile property, CD:1562-1563
- programming
 - core elements, 31
 - differences, server-side and client-side, 158-159
 - languages, development history, 5-6
 - modular, 17
 - object based, 10-16
 - OOP (object-oriented programming), 7-17
 - techniques, 85-106
- programs, code, 399
- project object, 1312-1313
- project.lock() method, 1313
- project.unlock() method, 1313-1314
- project.unwatch() method, 1314
- project.watch() method, 1315
- PROMPT (Windows environment variable), CD:1955
- prompt
 - boxes, prompt() method, code, 1147
 - method (Window object), 1086
 - property, CD:1600-1602
- prompt() method, code, 1147
- properties
 - \$1,\$2...\$9, 474
 - @set, 201-202
 - abbr, CD:1674-1675
 - ABORT, code, 720-721
 - above, 821-824

- ul style="list-style-type: none; padding-left: 0;">
- accept, CD:1585-1887
- acceptCharset, CD:1546-1548
- accessKey, CD:1480-1482,
 - CD:1497-1498, CD:1513-1514,
 - CD:1585-1588, CD:1602-1605
- action, 761, CD:1546-1548
- Add (Folders collection), CD:1909
- agent, 1327-1328
- align, 649, 662, 688, 995, 998
- aLink, code, CD:1507-1509
- alinkColor, code to set, 636-638, 1108
- all (Document object), 636
- alt, code, CD:1490-1491,
 - CD:1497-1499, CD:1577-1579,
 - CD:1585, CD:1589
- altKey, code, CD:1748-1749
- Anchor object, 564
- Anchor.name, 566-567
- Anchor.text, 567-568
- Anchor.x, 570-571
- Anchor.y, 571
- anchors, 636, 642-643, 1108, CD:1525
- appCodeName, code, 904-905
- Applet object, 572
- applets, 636, 644, 1108, CD:1525
- appName, code, 904-906
- appVersion, 904-907
- archive, CD:1490-1492, CD:1624-1626
- Area object, 573
- Area.hash, 578
- Area.host, 580
- Area.hostname, 582
- Area.href, 583
- Area.pathname, 587
- Area.port, 588
- Area.protocol, 589
- Area.search, 590
- Area.target, 591
- areas, 574-576, CD:1615-1616
- arguments, 353-355, CD:1930-1931,
 - CD:1942-1943, CD:2006
- Arguments, 243-247, CD:1926-1931.
 - CD:2005
- arity, 353, 356
- Array object, 249, 253-259
- arrays, code to access, 44
- ASPCode, 1183-1184
- ASPDescription, 1183-1184
- ASPErrors object, 1183
- ASPErrors.ASPCode, 1184
- ASPErrors.ASPDescription, 1184
- ASPErrors.Category, 1185
- ASPErrors.Column, 1185
- ASPErrors.Description, 1185
- ASPErrors.File, 1186
- ASPErrors.Line, 1186
- ASPErrors.Number, 1187
- ASPErrors.Source, 1187
- AtEndOfLine, CD:1912-1914
- AtEndOfStream, CD:1913-1915
- Attr object, CD:1404
- Attr.name, CD:1404
- Attr.ownerElement, CD:1406
- Attr.specified, CD:1404-1405
- Attr.value, CD:1405
- attrChange, CD:1756-1758
- Attributes, CD:1845-1847,
 - CD:1891-1893
- attributes, CD:1455-1456
- attrName, CD:1756-1759
- AvailableSpace, CD:1833-1834
- availHeight, 967-969
- availLeft, code to access, 969
- availTop, code to access, 970
- availWidth, code to access, 970
- axis, CD:1674-1677
- background, 821, 824-825,
 - CD:1507-1509
- backgroundColor, 649, 662, 688, 995,
 - 998-999
- backgroundImage, 649, 662, 688, 995,
 - 999-1000
- Bcc, 1355-1357
- below, 821, 825-826
- bgColor, 646, 821, 826-827, 1108
- bgcolor (Document object), 636
- Big Button object, 71
- blue, CD:1816
- BLUR, code, 721-722
- Body, 1355, 1358, CD:1528
- body (HTMLDocument object),
 - CD:1525
- Boolean object, 273-276
- border, code, 807-810, CD:1577-1579,
 - CD:1624-1626, CD:1688, CD:1691
- borderBottomWidth, 649, 662, 688, 995,
 - 1000
- borderColor, 649, 662, 688, 995, 1001
- borderLeftWidth, 649, 662, 688, 995,
 - 1002
- borderRightWidth, 649, 662, 688, 995,
 - 1002-1003

borderStyle, 649, 662, 688, 995, 1003
borderTopWidth, 649, 662, 688, 995, 1004
bottom, CD:1812-1813
bubbles, CD:1733-1735
Buffer, code, 1337
button, CD:1748-1750
Button object, 597
Button.form, 602
Button.name, 605
Button.type, 609
Button.value, 612
bvalue (HTMLTextAreaElement object), CD:1715
CacheControl, code, 1338
callee, 243-246, 353, 358
caller, 243, 247, 353, 358-359
cancelable, CD:1733-1736
caption, CD:1688-1689, CD:1691
Car object, code, 402
Category, 1183-1185
Cc, 1355, 1358
ceil (Math object), 408
cellIndex, CD:1674, CD:1678
cellPadding, CD:1688, CD:1692
cells, CD:1703-1705
cellSpacing, CD:1688, CD:1692-1693
ch, code, CD:1674, CD:1678, CD:1684-1685, CD:1703-1706, CD:1710-1711
CHANGE, code to access, 722-723, 744-745
Character, CD:1979-1980, CD:1407-1408
Charset, code, 1338, CD:1483, CD:1610, CD:1656
charset, CD:1480, CD:1609, CD:1655
Checkbox object, 615
Checkbox.checked, 618
Checkbox.defaultChecked, 621
Checkbox.form, 623-624
Checkbox.name, 626
Checkbox.type, 631
Checkbox.value, 634
checked, 615, 618-620, 938, 941, CD:1585, CD:1590
childNodes, CD:1455-1458
chOff, CD:1674, CD:1679, CD:1684-1686, CD:1703, CD:1706
cite, CD:1622-1623, CD:1653-1654
classes, 636, 649-650
className, CD:1539-1540
classname (Packages object), 460
clear, 649, 662, 688, 995, 1006, CD:1513
CLICK, code to access, 723-724
client object, 1194-1197
client.property, 1196
clientX, CD:1748-1750
clientY, CD:1748, CD:1751
clip.bottom, 821, 828-829
clip.height, 821, 829-830
clip.left, 821, 830-831
clip.right, 821, 831-832
clip.top, 821, 832-833
clip.width, 821, 833-834
closed, 1087, 1101-1102
code, CD:1430, CD:1490, CD:1624, CD:1627, CD:1741
codeBase, CD:1490, CD:1493, CD:1624, CD:1627-1628
CodePage, 1379-1380
codeType, CD:1624, CD:1628
color, 649, 662, 688, 995, 1007
colorDepth, code to access, 971
cols, CD:1560-1561, CD:1715-1717
colSpan, CD:1674, CD:1679
Column, 1183-1185, CD:1913, CD:1916
columnName, 1219-1222
compact, CD:1521-1522, CD:1524-1525, CD:1617-1618, CD:1636-1637, CD:1726
complete, 807, 810
ComputerName, CD:1964-1967
Connection.prototype, 1209-1210
constructor, code, 249, 253, 273-275, 289, 353
content, CD:1618-1619
contentDocument, CD:1554-1555, CD:1570-1571, CD:1624, CD:1629
ContentType, code, 1339
cookie, 636, 653, 1109, CD:1525, CD:1529
coords, code, CD:1480, CD:1497, CD:1483, CD:1499
Count, 1322, 1324, CD:1824-1826, CD:1843-1844, CD:1861-1863, CD:1909-1911, CD:1944-1945
Counter object, code, CD:1767-1768
Counter.identifier, CD:1768
Counter.listStyle, CD:1768-1769
Counter.separator, CD:1768-1769

- creating, code, 1209-1210, 1228, 1246-1247, 1267-1268, 1287
- crypto, 1087, 1103-1104
- CSSCharsetRule object, CD:1770
- CSSCharsetRule.encoding, CD:1770
- CSSFontFaceRule object, CD:1771
- CSSFontFaceRule.style, CD:1771-1772
- CSSImportRule object, CD:1772
- CSSImportRule.href, CD:1773
- CSSImportRule.media, CD:1773
- CSSImportRule.styleSheet, CD:1774
- CSSMediaRule object, CD:1775
- CSSMediaRule.cssRules, CD:1775
- CSSMediaRule.media, CD:1777
- CSSPageRule object, CD:1778
- CSSPageRule.selectorText, CD:1778
- CSSPageRule.style, CD:1779
- CSSPrimitiveValue object, CD:1781
- CSSPrimitiveValue.primitiveType, CD:1785
- cssRule (CSSStyleSheet object), CD:1801
- CSSRule object, CD:1787
- CSSRule.cssText, CD:1787-1788
- CSSRule.parentRule, CD:1788
- CSSRule.parentStyleSheet, CD:1789
- CSSRule.type, CD:1789
- CSSRuleList object, CD:1790
- CSSRuleList.length, CD:1791
- cssRules, CD:1775-1776, CD:1801-1802
- CSSStyleDeclaration object, CD:1792
- CSSStyleDeclaration.cssText, CD:1793
- CSSStyleDeclaration.length, CD:1796
- CSSStyleDeclaration.parentRule, CD:1797
- CSSStyleRule object, CD:1799
- CSSStyleRule.selectorText, CD:1799
- CSSStyleRule.style, CD:1800
- CSSStyleSheet object, CD:1801
- CSSStyleSheet.cssRules, CD:1801
- CSSStyleSheet.ownerRule, CD:1803
- cssText, CD:1787-1788, CD:1792-1794, CD:1805-1806
- CSSUnknownRule object, CD:1804
- CSSValue object, CD:1805
- CSSValue.cssText, CD:1805
- CSSValue.cssValueType, CD:1806
- CSSValueList object, CD:1807
- CSSValueList.length, CD:1808
- cssValueType, CD:1805-1806
- ctrlKey, CD:1748, CD:1751-1752
- current, 799-801, 1117
- CurrentDirectory, CD:1990-1993
- currentTarget, CD:1733, CD:1736
- Cursor object, 1219
- Cursor.columnName, 1222
- Cursor.prototype, 1228
- data, 698, CD:1407-1408, CD:1473, CD:1624
- database.prototype, 1246-1247
- Date object, 289, 307-308
- DateCreated, CD:1845, CD:1848-1849, CD:1891, CD:1894-1895
- DateLastAccessed, CD:1845, CD:1849-1850, CD:1891, CD:1895-1896
- DateLastModified, code, CD:1845, CD:1850-1851, CD:1891, CD:1896-1897
- dateTime, CD:1622-1623
- DBLCLICK, code to access, 724-725
- DbPool.prototype, 1267-1268
- declare, CD:1624, CD:1630
- defaultChecked, 615, 621-622, 938, 942-943, CD:1586, CD:1591
- defaultSelected, 915-916, CD:1641-1642
- defaultStatus, 1087, 1106
- defaultValue, 922-925, 1046, 1049, 1064-1067, CD:1586
- defer, CD:1655-1656
- description, 341, 899-900, 934-935, 1186, CD:1981
- Description (ASPErrors object), 1183, CD:1979, CD:2005
- detail, CD:1761-1762
- Dictionary object, CD:1824
- dir (HTMLElement object), CD:1539
- disabled, CD:1513-1515, CD:1586, CD:1592-1593, CD:1609-1611, CD:1639-1640
- display, 649, 662, 688, 995, 1008
- docType, CD:1412, CD:1420-1421
- document, 778, 781-784, 822, 834-835, 1087
- Document object, 636, CD:1412
- document.alinkColor, 637-638
- document.all, 638-639
- document.anchors, 641
- document.anchors.length, 642
- document.applets, 643

- document.applets.length, 644
- document.bgColor, 645
- document.classes, 648
- document.cookie, 652-653
- Document.docType, CD:1420
- Document.documentElement, CD:1421
- document.domain, 653
- document.embeds, 654
- document.embeds.length, 655
- document.fgColor, 655
- document.formName, 656
- document.forms, 657
- document.forms.length, 658
- document.ids, 661
- document.images, 664
- document.images.length, 665
- Document.implementation, CD:1423
- document.lastModified, 666
- document.layers, 667
- document.layers.length, 668
- document.linkColor, 669
- document.links, 670
- document.links.length, 671
- document.plugins, 681
- document.plugins.length, 682
- document.referrer, 682
- document.tags, 687
- document.title, 690
- document.URL, 692
- document.vlinkColor, 692
- documentElement, CD:1412, CD:1421
- DocumentType object, CD:1425
- DocumentType.entities, CD:1426
- DocumentType.internalSubset, CD:1426
- DocumentType.name, CD:1427
- DocumentTypenotations,
CD:1427-1428
- DocumentType.publicId, CD:1428
- DocumentType.systemId, CD:1429
- domain, code, 636, 1109, CD:1525,
CD:1530
- DOMException object, CD:1429-1430
- DOMException.code, CD:1430
- DRAGDROP, code, 726
- Drive, CD:1833, CD:1845,
CD:1852-1853, CD:1898
- DriveLetter, CD:1833-1835
- Drives collection, CD:1843-1844
- DriveType, CD:1833-1836
- E, code, 416-417
- E (Math object), 408
- Element object, CD:1434
- element.style.visibility, 152
- Element.tagName, CD:1444
- ElementCSSInlineStyle.style, CD:1812
- elements, code, 761, CD:1546, CD:1549
- elements.length, 761, 765-766
- embeds, 636, 654, 1109
- enabledPlugin, code to access, 899-901
- encoding, 761, CD:1770-1771
- enctype, code, CD:1546, CD:1550
- entities, CD:1425-1426
- Entity object, CD:1445
- Entity.notationName, CD:1445
- Entity.publicId, CD:1446
- Entity.systemId, CD:1446
- Environment, CD:1990, CD:1993
- Error, 341-342, CD:1972, CD:1975
- ERROR, code to access, 727
- Error.description, 342
- Error.message, 343
- Error.name, 343-344
- Error.number, 344-345
- Errorsto, 1355, 1361
- event (HTMLScriptElement object),
CD:1655
- Event object, 698, CD:1733
- Event.ABORT, 720
- Event.BLUR, 721
- Event.bubbles, CD:1734-1735
- Event.cancelable, CD:1735
- Event.CHANGE, 722
- Event.CLICK, 723
- Event.currentTarget, CD:1736
- event.data, 701-702
- Event.DBLCCLICK, 724
- Event.DRAGDROP, 725
- Event.ERROR, 726-727
- Event.eventPhase, CD:1736-1737
- Event.FOCUS, 728
- event.height, 702-703
- Event.KEYDOWN, 729
- Event.KEYPRESS, 730
- Event.KEYUP, 731
- event.layerX, 703-704
- event.layerY, 704
- Event.LOAD, 732
- event.modifiers, 705-706
- Event.MOUSEDOWN, 733
- Event.MOUSEMOVE, 734
- Event.MOUSEOUT, 735-736
- Event.MOUSEOVER, 736

- Event.MOUSEUP, 737
- Event.MOVE, 738
- event.pageX, 707
- event.pageY, 708
- Event.RESET, 739
- Event.RESIZE, 741
- event.screenX, 709-710
- event.screenY, 711
- Event.SELECT, 742
- Event.SUBMIT, 743
- event.target, 712, CD:1739
- Event.timeStamp, CD:1740
- event.type, 713, CD:1734
- Event.UNLOAD, 744
- event.which, 718
- event.width, 718
- EventException object, CD:1741
- EventException.code, CD:1742
- eventPhase, code, CD:1733, CD:1737
- Expires, code, 1341
- ExpiresAbsolute, code, 1341
- face, CD:1505-1506, CD:1544-1545
- fgColor, 656, 1109
- fgcolor (Document object), 636
- File, code, 1183, 1186, CD:1845
- File.constructor, 1279-1280
- File.prototype, 1286-1287
- filename, 934-936
- Files, CD:1861, CD:1891, CD:1899
- FileSystem, CD:1833, CD:1836-1837, CD:1864
- FileUpload object, 745
- FileUpload.form, 749-750
- FileUpload.name, 752-753
- FileUpload.type, 757
- FileUpload.value, 759
- firstChild, CD:1455, CD:1459
- FOCUS, code to access, 728-729
- Folder object, CD:1891
- Folder.Attributes, CD:1892-1893
- Folder.DateCreated, CD:1894
- Folder.DateLastAccessed, CD:1895
- Folder.DateLastModified, CD:1896
- Folder.Drive, CD:1898
- Folder.Files, CD:1899
- Folder.IsRootFolder, CD:1899
- Folder.Name, CD:1901
- Folder.ParentFolder, CD:1902
- Folder.Path, CD:1903
- Folder.ShortName, CD:1903-1904
- Folder.ShortPath, CD:1904-1905
- Folder.Size, CD:1905-1906
- Folder.SubFolders, CD:1906-1907
- Folder.Type, CD:1907
- Folders collection, CD:1909
- Folders.Count, CD:1910
- Folders.Item, CD:1911
- fontFamily, 649, 662, 688, 995, 1008-1009
- fontSize, 649, 662, 688, 995, 1010
- fontStyle, 649, 662, 688, 995, 1010-1011
- fontWeight, 649, 662, 688, 995, 1011-1012
- form, 602-603, 624, 794, 926, 944
 - Button object, 597
 - Checkbox object, 615
 - FileUpload object, 745
 - Hidden object, 793
 - HTMLButtonElement object, CD:1513
 - HTMLInputElement object, CD:1586
 - HTMLIsIndexElement object, CD:1600
 - HTMLSelectElement object, CD:1659
 - HTMLTextAreaElement object, CD:1715
 - Password object, 922
 - property, forms, code to access, 1035-1036
 - Radio object, 938
 - Reset object, 952
 - Select object, 975
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1064
- Form object, 761
- Form.action, 763
- Form.elements, 764
- Form.elements.length, 765
- Form.encoding, 766-767
- Form.length, 768-769
- Form.method, 769
- Form.name, 770
- Form.onReset, 771
- Form.OnSubmit, 772
- Form.target, 775
- formKey, 1327-1329
- formName (window.document property), 1109
- forms (Document object), 636, 657-658, 1109, CD:1525

frame, CD:1688, CD:1697
Frame object, 778
Frame.document, 781
Frame.frames, 782-783
Frame.length, 783
Frame.name, 784
Frame.parent, 787
Frame.self, 788
Frame.top, 790
Frame.window, 792
frameBorder, CD:1554-1556,
 CD:1570-1572
frames, 783, 778, 1087, 1114
FreeSpace, CD:1833, CD:1837-1838
From, 1355, 1362
FullName, CD:1929, CD:1934,
 CD:2005-2007, CD:2020-2021
Function object, 353
Function.arguments, 354-355
Function.arity, 355-356
Function.callee, 357
Function.caller, 358-359
Function.constructor, 359
Function.length, 360
Function.prototype, 360-361
getPropertyPriority() method, code,
 CD:1795
getPropertyValue() method, code,
 CD:1795-1796
getting, code, CD:1805
global, 467, 478
Global object, 365
Global.undefined, 380
green, CD:1816-1817
handleEvent (Link object), 865
hash, 573, 579-580, 865-868, 885-887
headers, CD:1674, CD:1680
height, 698, 807, 811-812, 971-972,
 CD:1490
Hidden object, 793
Hidden.form, 794
Hidden.name, 795
Hidden.type, 795
Hidden.value, 797
history, 1087, 1117-1118
History object, 799
History.current, 800
History.length, 802-803
History.next, 803
History.previous, 804
host, 573, 581, 865, 868-869, 885
hostname, code, 573, 582-583, 865, 885,
 889-890
Hotkey, CD:2005, CD:2008
href, 573, 583-584, 865, 886, 890-891,
 CD:1480
hrefLang, code, CD:1480, CD:1485,
 CD:1609, CD:1612
hspace, code, 807, 812-813, CD:1490,
 CD:1494, CD:1577
HTMLAnchorElement object, CD:1480
HTMLAnchorElement.accessKey,
 CD:1481
HTMLAnchorElement.charset, CD:1482
HTMLAnchorElement.coords, CD:1483
HTMLAnchorElement.href, CD:1484
HTMLAnchorElement.hrefLang,
 CD:1485
HTMLAnchorElement.name, CD:1485
HTMLAnchorElement.rel, CD:1486
HTMLAnchorElement.rev,
 CD:1486-1487
HTMLAnchorElement.shape, CD:1487
HTMLAnchorElement.tabIndex,
 CD:1488
HTMLAnchorElement.target, CD:1488
HTMLAnchorElement.type, CD:1489
HTMLAppletElement object, CD:1490
HTMLAppletElement.align, CD:1490
HTMLAppletElement.alt, CD:1491
HTMLAppletElement.archive, CD:1492
HTMLAppletElement.code, CD:1492
HTMLAppletElement.codeBase,
 CD:1493
HTMLAppletElement.height, CD:1493
HTMLAppletElement.hspace, CD:1494
HTMLAppletElement.name, CD:1495
HTMLAppletElement.object, CD:1495
HTMLAppletElement.vspace, CD:1496
HTMLAppletElement.width, CD:1496
HTMLAreaElement object, CD:1497
HTMLAreaElement.accessKey,
 CD:1498
HTMLAreaElement.alt, CD:1498
HTMLAreaElement.coords, CD:1499
HTMLAreaElement.href, CD:1500
HTMLAreaElement.noHref, CD:1500
HTMLAreaElement.shape, CD:1501
HTMLAreaElement.tagIndex, CD:1501
HTMLAreaElement.target, CD:1502
HTMLBaseElement object, CD:1503
HTMLBaseElement.href,
 CD:1503-1504

HTMLBaseElement.target, CD:1504	HTMLElement.dir, CD:1540
HTMLBaseFontElement object, CD:1505	HTMLElement.id, CD:1541
HTMLBaseFontElement.color, CD:1505-1506	HTMLElement.lang, CD:1541-1542
HTMLBaseFontElement.face, CD:1506	HTMLElement.title, CD:1542
HTMLBaseFontElement.size, CD:1507	HTMLFieldSetElement object, CD:1543
HTMLBodyElement object, CD:1507	HTMLFieldSetElement.form, CD:1543
HTMLBodyElement.aLink, CD:1508	HTMLFontElement object, CD:1544
HTMLBodyElement.background, CD:1509	HTMLFontElement.color, CD:1545
HTMLBodyElement.bgColor, CD:1509	HTMLFontElement.face, CD:1545
HTMLBodyElement.link, CD:1510	HTMLFontElement.size, CD:1546
HTMLBodyElement.text, CD:1511	htmlFor, code, CD:1602-1604, CD:1655
HTMLBodyElement.vLink, CD:1511	HTMLFormElement object, CD:1546-1547
HTMLBRElement.clear, CD:1512-1513	HTMLFormElement.acceptCharset, CD:1547
HTMLButtonElement object, CD:1513	HTMLFormElement.action, CD:1548
HTMLButtonElement.accessKey, CD:1514	HTMLFormElement.elements, CD:1549
HTMLButtonElement.disabled, CD:1514-1515	HTMLFormElement.enctype, CD:1549
HTMLButtonElement.form, CD:1515	HTMLFormElement.length, CD:1550
HTMLButtonElement.name, CD:1516	HTMLFormElement.method, CD:1551
HTMLButtonElement.tabIndex, CD:1516	HTMLFormElement.name, CD:1551
HTMLButtonElement.type, CD:1517	HTMLFormElement.target, CD:1553
HTMLButtonElement.value, CD:1517	HTMLFrameElement object, CD:1554
HTMLCollection object, CD:1518	HTMLFrameElement.contentDocument, CD:1555
HTMLCollection.length, CD:1520	HTMLFrameElement.frameBorder, CD:1555
HTMLDirectoryElement object, CD:1521	HTMLFrameElement.longDesc, CD:1556
HTMLDirectoryElement.compact, CD:1522	HTMLFrameElement.marginHeight, CD:1557
HTMLDivElement object, CD:1523	HTMLFrameElement.marginWidth, CD:1557
HTMLDivElement.align, CD:1523	HTMLFrameElement.name, CD:1558
HTMLDListElement object, CD:1524	HTMLFrameElement.noResize, CD:1558-1559
HTMLDListElement.compact, CD:1524	HTMLFrameElement.scrolling, CD:1559
HTMLDocument object, CD:1525	HTMLFrameElement.src, CD:1560
HTMLDocument.anchors, CD:1526	HTMLFrameSetElement object, CD:1560
HTMLDocument.applets, CD:1527	HTMLFrameSetElement.cols, CD:1561-1562
HTMLDocument.body, CD:1528	HTMLHeadElement object, CD:1562
HTMLDocument.cookie, CD:1529	HTMLHeadElement.profile, CD:1563
HTMLDocument.domain, CD:1530	HTMLHeadingElement object, CD:1564
HTMLDocument.forms, CD:1530	HTMLHeadingElement.align, CD:1564
HTMLDocument.images, CD:1532	HTMLHRElement object, CD:1565
HTMLDocument.links, CD:1533	HTMLHRElement.align, CD:1566
HTMLDocument.referrer, CD:1534	HTMLHRElement.noShade, CD:1566
HTMLDocument.title, CD:1535	HTMLHRElement.size, CD:1567
HTMLDocument.URL, CD:1535	
HTMLElement object, CD:1539	
HTMLElement.className, CD:1540	

HTMLHRElement.width, CD:1567
HTMLHtmlElement object, CD:1568
HTMLHtmlElement.version, CD:1569
HTMLIFrameElement object, CD:1570
HTMLIFrameElement.align, CD:1570
HTMLIFrameElement.contentDocument,
CD:1571
HTMLIFrameElement.frameBorder,
CD:1571
HTMLIFrameElement.height, CD:1572
HTMLIFrameElement.longDesc,
CD:1573
HTMLIFrameElement.marginHeight,
CD:1573
HTMLIFrameElement.marginWidth,
CD:1574
HTMLIFrameElement.name,
CD:1574-1575
HTMLIFrameElement.scrolling,
CD:1575
HTMLIFrameElement.src, CD:1576
HTMLIFrameElement.width, CD:1576
HTMLImageElement object, CD:1577
HTMLImageElement.align, CD:1578
HTMLImageElement.alt, CD:1578-1579
HTMLImageElement.border, CD:1579
HTMLImageElement.height, CD:1580
HTMLImageElement.hspace, CD:1580
HTMLImageElement.isMap, CD:1581
HTMLImageElement.longDesc,
CD:1581-1582
HTMLImageElement.lowSrc, CD:1582
HTMLImageElement.src, CD:1583
HTMLImageElement.useMap, CD:1583
HTMLImageElement.vspace, CD:1584
HTMLImageElement.width, CD:1584
HTMLInputElement object, CD:1585
HTMLInputElement.accept, CD:1587
HTMLInputElement.accessKey,
CD:1587
HTMLInputElement.align, CD:1588
HTMLInputElement.alt, CD:1588
HTMLInputElement.checked,
CD:1589-1590
HTMLInputElement.defaultChecked,
CD:1591
HTMLInputElement.defaultValue,
CD:1591-1592
HTMLInputElement.disabled, CD:1592
HTMLInputElement.form, CD:1593

HTMLInputElement.maxLength,
CD:1594
HTMLInputElement.name, CD:1594
HTMLInputElement.readOnly, CD:1595
HTMLInputElement.size, CD:1596
HTMLInputElement.src, CD:1597
HTMLInputElement.tabIndex, CD:1597
HTMLInputElement.type, CD:1598
HTMLInputElement.useMap, CD:1599
HTMLInputElement.value, CD:1599
HTMLIsIndexElement object, CD:1600
HTMLIsIndexElement.form, CD:1601
HTMLIsIndexElement.prompt, CD:1601
HTMLLabelElement object, CD:1602
HTMLLabelElement.accessKey,
CD:1603
HTMLLabelElement.form, CD:1603
HTMLLabelElement.htmlFor, CD:1604
HTMLLegendElement object, CD:1605
HTMLLegendElement.accessKey,
CD:1605
HTMLLegendElement.align, CD:1606
HTMLLegendElement.form, CD:1606
HTMLLIElement object, CD:1607
HTMLLIElement.type, CD:1608
HTMLLIElement.value, CD:1608
HTMLLinkElement object, CD:1609
HTMLLinkElement.charset, CD:1610
HTMLLinkElement.disabled,
CD:1610-1611
HTMLLinkElement.href, CD:1611
HTMLLinkElement.hreflang, CD:1612
HTMLLinkElement.media, CD:1612
HTMLLinkElement.rel, CD:1613
HTMLLinkElement.rev, CD:1613-1614
HTMLLinkElement.target, CD:1614
HTMLLinkElement.type, CD:1615
HTMLMapElement object, CD:1615
HTMLMapElement areas, CD:1616
HTMLMapElement.name,
CD:1616-1617
HTMLMenuElement object, CD:1617
HTMLMenuElement.compact, CD:1618
HTMLMetaElement object, CD:1618
HTMLMetaElement.content, CD:1619
HTMLMetaElement.httpEquiv,
CD:1620
HTMLMetaElement.name, CD:1620
HTMLMetaElement.scheme, CD:1621
HTMLModElement object, CD:1622
HTMLModElement.cite, CD:1622

HTMLModElement.dateTime, CD:1623	HTMLParagraphElement.align,
HTMLObjectElement object, CD:1624	CD:1648
HTMLObjectElement.align, CD:1625,	HTMLParamElement object, CD:1649
CD:1629	HTMLParamElement.name,
HTMLObjectElement.archive, CD:1625	CD:1649-1650
HTMLObjectElement.border, CD:1626	HTMLParamElement.type, CD:1650
HTMLObjectElement.code, CD:1627	HTMLParamElement.value, CD:1651
HTMLObjectElement.codeBase,	HTMLParamElement.valueType,
CD:1627	CD:1651
HTMLObjectElement.codeType,	HTMLPreElement object, CD:1652
CD:1628	HTMLPreElement.width, CD:1653
HTMLObjectElement.contentDocument,	HTMLQuoteElement object, CD:1653
CD:1628-1629	HTMLQuoteElement.cite, CD:1654
HTMLObjectElement.declare, CD:1630	HTMLScriptElement object, CD:1655
HTMLObjectElement.form, CD:1630	HTMLScriptElement.charset, CD:1655
HTMLObjectElement.height, CD:1631	HTMLScriptElement.defer, CD:1656
HTMLObjectElement.hspace,	HTMLScriptElement.event, CD:1657
CD:1631-1632	HTMLScriptElement.htmlFor, CD:1657
HTMLObjectElement.name, CD:1632	HTMLScriptElement.src, CD:1657
HTMLObjectElement.standby, CD:1633	HTMLScriptElement.text, CD:1658
HTMLObjectElement.tabIndex,	HTMLScriptElement.type, CD:1658
CD:1633	HTMLSelectElement object, CD:1659
HTMLObjectElement.type, CD:1634	HTMLSelectElement.disabled, CD:1661
HTMLObjectElement.useMap,	HTMLSelectElement.form, CD:1663
CD:1634-1635	HTMLSelectElement.length, CD:1663
HTMLObjectElement.vspace, CD:1635	HTMLSelectElement.multiple, CD:1664
HTMLObjectElement.width, CD:1636	HTMLSelectElement.name, CD:1665
HTMLListElement object, CD:1636	HTMLSelectElement.options, CD:1665
HTMLListElement.compact, CD:1637	HTMLSelectElement.selectedIndex,
HTMLListElement.start,	CD:1667
CD:1637-1638	HTMLSelectElement.size,
HTMLListElement.type, CD:1638	CD:1667-1668
HTMLOptGroupElement object,	HTMLSelectElement.tabIndex,
CD:1639	CD:1668
HTMLOptGroupElement.disabled,	HTMLSelectElement.type, CD:1669
CD:1639	HTMLSelectElement.value,
HTMLOptGroupElement.label,	CD:1669-1670
CD:1640	HTMLStyleElement object, CD:1670
HTMLOptionElement object, CD:1641	HTMLStyleElement.disabled, CD:1671
HTMLOptionElement.defaultSelected,	HTMLStyleElement.media,
CD:1642	CD:1671-1672
HTMLOptionElement.disabled,	HTMLStyleElement.type, CD:1672
CD:1642-1643	HTMLTableCaptionElement object,
HTMLOptionElement.form, CD:1643	CD:1673
HTMLOptionElement.index, CD:1644	HTMLTableCaptionElement.align,
HTMLOptionElement.label, CD:1645	CD:1673
HTMLOptionElement.selected,	HTMLTableCellElement object,
CD:1645	CD:1674
HTMLOptionElement.text, CD:1646	HTMLTableCellElement.abbr, CD:1675
HTMLOptionElement.value, CD:1647	HTMLTableCellElement.align, CD:1676
HTMLParagraphElement object,	HTMLTableCellElement.axis, CD:1676
CD:1648	

HTMLTableCellElement.bgColor,
CD:1677
HTMLTableCellElement.cellIndex,
CD:1677
HTMLTableCellElement.ch, CD:1678
HTMLTableCellElement.chOff,
CD:1678-1679
HTMLTableCellElement.colSpan,
CD:1679
HTMLTableCellElement.headers,
CD:1680
HTMLTableCellElement.height,
CD:1680
HTMLTableCellElement.noWrap,
CD:1681
HTMLTableCellElement.rowSpan,
CD:1681
HTMLTableCellElement.scope,
CD:1682
HTMLTableCellElement.vAlign,
CD:1682
HTMLTableCellElement.width,
CD:1683
HTMLTableColElement object,
CD:1684
HTMLTableColElement.align, CD:1684
HTMLTableColElement.ch, CD:1685
HTMLTableColElement.chOff, CD:1686
HTMLTableColElement.span, CD:1686
HTMLTableColElement.vAlign,
CD:1687
HTMLTableColElement.width,
CD:1687-1688
HTMLTableElement object, CD:1688
HTMLTableElement.align, CD:1689
HTMLTableElement.bgColor, CD:1690
HTMLTableElement.border, CD:1690
HTMLTableElement.caption, CD:1691
HTMLTableElement.cellPadding,
CD:1692
HTMLTableElement.cellSpacing,
CD:1692
HTMLTableElement.frame, CD:1697
HTMLTableElement.rows, CD:1698
HTMLTableElement.rules, CD:1699
HTMLTableElement.summary, CD:1699
HTMLTableElement.tBodies, CD:1700
HTMLTableElement.tFoot, CD:1701
HTMLTableElement.tHead, CD:1701
HTMLTableElement.width, CD:1702

HTMLTableRowElement object,
CD:1703
HTMLTableRowElement.align,
CD:1703
HTMLTableRowElement.bgColor,
CD:1704
HTMLTableRowElement.cells, CD:1705
HTMLTableRowElement.ch, CD:1705
HTMLTableRowElement.chOff,
CD:1706
HTMLTableRowElement.rowIndex,
CD:1708
HTMLTableRowElement.sectionRowIndex,
CD:1708
HTMLTableRowElement.vAlign,
CD:1709
HTMLTableSectionElement object,
CD:1710
HTMLTableSectionElement.align,
CD:1710
HTMLTableSectionElement.ch,
CD:1711
HTMLTableSectionElement.chOff,
CD:1711
HTMLTableSectionElement.rows,
CD:1713
HTMLTableSectionElement.vAlign,
CD:1714
HTMLTextAreaElement object,
CD:1715
HTMLTextAreaElement.accessKey,
CD:1715-1716
HTMLTextAreaElement.cols, CD:1717
HTMLTextAreaElement.defaultValue,
CD:1717
HTMLTextAreaElement.disabled,
CD:1718
HTMLTextAreaElement.form, CD:1719
HTMLTextAreaElement.name, CD:1720
HTMLTextAreaElement.readOnly,
CD:1720
HTMLTextAreaElement.rows, CD:1721
HTMLTextAreaElement.tabIndex,
CD:1722
HTMLTextAreaElement.type, CD:1723
HTMLTextAreaElement.value, CD:1723
HTMLTitleElement object, CD:1724
HTMLTitleElement.text, CD:1725
HTMLULListElement object, CD:1726
HTMLULListElement.compact, CD:1726
HTMLULListElement.type, CD:1727

- httpEquiv, CD:1618-1620
- IconLocation, CD:2005, CD:2009
- id (HTMLElement object), CD:1539
- identifier, CD:1767-1768
- ids, 662-663, 636
- ignoreCase, 467, 479
- Image object, 807
- Image.border, 809
- Image.complete, 810
- Image.height, 811
- Image.hspace, 812
- Image.lowsrc, 813
- Image.name, 814
- Image.src, 817
- Image.vspace, 819
- Image.width, 820
- images, 636, 664-665, 1109, CD:1525
- imageX, 1327, 1330
- imageY, 1327, 1330
- implementation, CD:1413, CD:1423-1424
- index, 249, 253-254, CD:1641, CD:1644
- Infinity, code, 365, 373-374
- innerHeight, 1087, 1119
- innerWidth, 1087, 1120-1121
- input, 250, 254, 467, 480
- Interactive, CD:1929, CD:1936
- internalSubset, CD:1425-1427
- ip, 1327, 1331
- IsClientConnected, code, 1342
- isMap, CD:1577, CD:1581
- IsReady, CD:1833, CD:1838
- IsRootFolder, CD:1891, CD:1900
- Item, CD:1825, CD:1843, CD:1861-1864, CD:1909-1912, CD:1960-1963
- Item(), CD:1827-1828, CD:1844-1845
- java (Packages object), 460
- javaArray.length, 391
- Key (Dictionary object), CD:1825
- Key() (Dictionary object), CD:1829-1830
- KEYDOWN, code to access, 729-730
- KEYUP, code to access, 731-732
- label, CD:1639-1641, CD:1645
- lang, CD:1539, CD:1542
- language, 904, 908
- lastChild, CD:1455, CD:1462
- lastIndex, 250, 255-256, 467, 481
- lastMatch, 467, 482
- lastModified, prices, 636, 666-667, 1109
- lastParen, code, 467, 482-483
- Layer object, 821
- Layer.above, 823
- Layer.background, 824
- Layer.below, 825
- Layer.bgColor, 826
- Layer.clip.bottom, 828
- Layer.clip.height, 829
- Layer.clip.left, 830
- Layer.clip.right, 831
- Layer.clip.top, 832
- Layer.clip.width, 833
- Layer.document, 834
- Layer.left, 837
- Layer.name, 843
- Layer.pageX, 849
- Layer.pageY, 850
- Layer.parentLayer, 851
- Layer.siblingAbove, 856-857
- Layer.siblingBelow, 857
- Layer.src, 858
- Layer.top, 859
- Layer.visibility, 861
- Layer.zIndex, 864
- layers, 636, 667-668, 1109
- layerX (Event object), 698
- layerY, code to change, 704-705
- layerY (Event object), 698
- LCID, 1379, 1383
- left, 822, 837, CD:1812-1814
- leftContext, 467, 483-484
- length, 116-119, 360, 390-392, 645, 655
 - Arguments object, 243, 247-249
 - Array object, 250, 256-257
 - CSSValueList object, CD:1807
 - Form object, 761
 - Frame object, 778
 - Function object, 353
 - History object, 799
 - HTMLCollection object, CD:1518
 - HTMLFormElement object, CD:1546
 - HTMLSelectElement object, CD:1659
 - JSObject object, 397
 - Plugin object, 934
 - Select object, 975
 - String object, 500
 - Window object, 1087
 - WshSpecialFolders object, CD:2013
 - WshUnnamed object, CD:2016

Length, CD:1944, CD:1954, CD:1960
Line, code, 1183, 1186, CD:1913,
 CD:1916-1917, CD:1979-1982
lineHeight, code, 649, 662, 688, 995,
 1012-1013
lineStyleType (Style object), 995
link, code, 670-671, CD:1507, CD:1510
Link object, 865
Link.hash, 867
Link.host, 868
Link.hostname, 869
Link.href, 870
Link.pathname, 877-878
Link.port, 878-879
Link.protocol, 879
Link.search, 880
Link.target, 881
Link.text, 882
linkColor, link colors, code to set,
 669-670
linkColor (Document object), 636, 1109
links (Document object), 636, 1109,
 CD:1525
listStyle, CD:1767-1769
listStyleType, code, 649, 662, 688,
 013-1014
LN2, 408, 420
LN10, code, 408, 419
LOAD, code, 733
localName, CD:1455, CD:1462-1463
location, code, 1087, 1122
Location object, 885
Location.hash, 886
Location.host, 887-888
Location.hostname, 889
Location.href, 890
Location.pathname, 891
Location.port, 892
Location.protocol, 893
Location.search, 895
locationbar, code, 1087, 1122-1123
Lock.constructor, 1301
Lock.prototype, 1303-1304
LOG2E, 408, 423
LOG10E, 408, 422
longDesc, CD:1554-1556, CD:1570,
 CD:1573, CD:1577, CD:1582
lowsrc, 807, 813, CD:1577-1578,
 CD:1582
lowSrc (HTMLImageElement object),
 CD:1577
marginBottom, 649, 662, 688, 996,
 1014-1015
marginHeight, CD:1554, CD:1557,
 CD:1570, CD:1573-1574
marginLeft, 649, 662, 688, 996, 1015
marginRight, 649, 662, 688, 996, 1016
marginTop, 649, 662, 688, 996, 1018
marginWidth, code, CD:1554,
 CD:1557-1558, CD:1570, CD:1574
Math object, 408
Math.E, 416
Math.LN2, 420
Math.LN10, 419
Math.LOG2E, 422
Math.LOG10E, 421-422
Math.SQRT1_2, 431
Math.SQRT2, 432
MAX VALUE, code, 441-442
MAX VALUE (Number object), 439
maxLength, code, CD:1586, CD:1594
media, CD:1609, CD:1612-1613,
 CD:1670-1672, CD:1772-1775,
 CD:1777
menubar, code, 1087, 1124-1125
message (Error object), 342
metaKey, code, CD:1748, CD:1753
method, 761, 769-770, 1327, 1331-1332,
 CD:1547
MimeType object, 899
MimeType.description, 900
MimeType.enabledPlugin, 901
MimeType.suffixes, 901
MimeType.type, 902
mimeTypes, 904, 909
MIN VALUE, 439, 442
modifiers, 698, 06-707
MOUSEDOWN, code to access, 734
MouseEvent object, CD:1748, CD:1756
MouseEvent.altKey, CD:1749
MouseEvent.button, CD:1749
MouseEvent.clientX, CD:1750
MouseEvent.clientY, CD:1751
MouseEvent.ctrlKey, CD:1751
MouseEvent.metaKey, CD:1753
MouseEvent.relatedTarget, CD:1753
MouseEvent.screenX, CD:1754
MouseEvent.screenY, CD:1754-1755
MouseEvent.shiftKey, CD:1755
MOUSEMOVE, code to access, 735
MOUSEOUT, code, 736
MOUSEOVER, code to access, 736-737

MOUSEUP, code to access, 737-738
 MOVE, code to access, 738-739
 multiline, 467, 484-485
 multiple, CD:1659, CD:1664
 MutationEvent object, CD:1756
 MutationEvent.attrChange, CD:1758
 MutationEvent.attrName, CD:1758
 MutationEvent.newValue, CD:1760
 MutationEvent.prevValue, CD:1760
 MutationEvent.relatedNode, CD:1757
 name, 605-606, 626-627, 770-771, 795,
 814

- Anchor object, 564
- Attr object, CD:1404
- Button object, 597
- Checkbox object, 615
- DocumentType object, CD:1425
- Error object, 342
- FileUpload object, 745
- Form object, 761
- Frame object, 778
- Hidden object, 793
- HTMLAnchorElement object,
 CD:1480
- HTMLAppletElement object,
 CD:1490
- HTMLButtonElement object,
 CD:1513
- HTMLFormElement object,
 CD:1547
- HTMLFrameElement object,
 CD:1554
- HTMLIFrameElement object,
 CD:1570
- HTMLImageElement object,
 CD:1577
- HTMLInputElement object,
 CD:1586
- HTMLMapElement object,
 CD:1615
- HTMLMetaElement object,
 CD:1618
- HTMLObjectElement object,
 CD:1624
- HTMLParamElement object,
 CD:1649
- HTMLSelectElement object,
 CD:1659
- HTMLTextAreaElement object,
 CD:1715
- Image object, 807

- Layer object, 822
- Password object, 922
- Plugin object, 934
- property, 1037-1038, 1054,
 1071-1072
- Radio object, 938
- Reset object, 952
- Select object, 975
- Submit object, 1030
- Text object, 1046
- Textarea object, 1064
- Window object, 1087
- Name, CD:1845, CD:1854, CD:1891,
 CD:1929, CD:1936
- Named, CD:1944, CD:1948-1949
- NamedNodeMap.length, CD:1450
- namespaceURI, CD:1455, CD:1463
- NaN, code, 365, 439, 443
- navigator object, 904
- navigator.appCodeName, 905
- navigator.appName, 906
- navigator.appVersion, 906
- navigator.language, 908
- navigator.mimeTypes, 908-909
- navigator.platform, 909
- navigator.plugins, 910
- navigator.userAgent, 913
- NEGATIVE INFINITY, 440, 444
- netscape (Packages object), 460
- newValue, CD:1756, CD:1760-1761
- next, 799, 803-804, 1117
- nextSibling, code to read, CD:1455,
 CD:1464
- Node object, CD:1455
- Node.attributes, CD:1456
- Node.childNodes, CD:1457
- Node.firstChild, CD:1458-1459
- Node.lastChild, CD:1462
- Node.localName, CD:1462
- Node.namespaceURI, CD:1463
- Node.nextSibling, CD:1463-1464
- Node.nodeName, CD:1464
- Node.nodeType, CD:1465
- Node.nodeValue, CD:1465
- Node.ownerDocument, CD:1466-1467
- Node.parentNode, CD:1467
- Node.prefix, CD:1468
- Node.previousSibling, CD:1468
- nodeName, CD:1455, CD:1464
- nodeType, CD:1455, CD:1465
- nodeValue, CD:1455, CD:1465-1466

noHref, CD:1497, CD:1500-1501
noResize, CD:1554, CD:1559
noShade, CD:1565-1567
Notation object, CD:1470
Notation.publicId, CD:1471
Notation.systemId, CD:1471
notationName, CD:1445-1446
notations, CD:1425, CD:1428
noWrap, CD:1674, CD:1681
number (Error object, 342
Number, 365, 439, 1183, 1187, CD:1979
Number.constructor, 440-441
Number.MAX_VALUE, 441
Number.MIN_VALUE, 442
Number.NaN, 443
Number.NEGATIVE_INFINITY, 443
Number.POSITIVE_INFINITY, 444
Number.prototype, 445
Object object, 452
Object.constructor, 453
Object.prototype, 455
objects, 19, CD:1490, CD:1495-1496
of instances, code to display, 975-976,
1030-1031, 1064-1065
offscreenBuffering, 1087, 1129
onBlur, 745, 778, 822
onChange (FileUpload object), 745
onClick (Link object), 865
onclick, 93
onDbClick (Link object), 865
onFocus, 745, 778, 822
onKeyDown (Link object), 865
onKeyPress (Link object), 865
onLoad (Layer object), 822
onMouseDown (Link object), 865
onMouseOut, 822, 865
onMouseOver, 822, 865
onMouseUp (Link object), 865
onMove (Frame object), 778
onReset (Form object), 761
onResize (Frame object), 778
onSubmit (Form object), 761
opener, 1087, 1138-1139
Option object, 915
Option.defaultSelected, 916
Option.selected, 917
Option.text, 918
Option.value, 920
options, 975, 988-989, CD:1659,
CD:1665-1666
Organization, 1355, 1362
outerHeight, 1087, 1139-1140
outerWidth, 1087, 1141
ownerDocument, CD:1455, CD:1467
ownerElement, CD:1404-1406
ownerRule, CD:1801-1803
Package object, 460
Packages.className, 461
Packages.java, 461-462
Packages.netscape, 462
Packages.sun, 463
paddingBottom, 649, 663, 688, 996,
1019
paddingLeft, 649, 663, 688, 996,
1019-1020
paddingRight, 649, 663, 688, 996,
1020-1021
paddingTop, 649, 663, 688, 996,
1022-1023
pageX, 698, 707-708, 822, 849-850
pageXOffset, 1088, 1142-1143
pageY, 709, 698, 822, 850-851
pageYOffset, 1088, 1143-1144
parent, 778, 1088, 1144
ParentFolder, CD:1845, CD:1856,
CD:1902
parentLayer, 822, 851-852
parentNode, CD:1455, CD:1467
parentRule, CD:1787-1788, CD:1792,
CD:1797
parentStyleSheet, CD:1787-1789
Password object, 922
Password.defaultValue, 924
Password.form, 925-926
Password.name, 927
Password.type, 930
Password.value, 932
Path, CD:1833, CD:1838-1839,
CD:1845, CD:1857
pathname, 573, 587-588, 865, 886,
891-892
personalbar, visible property, 1088,
1145-1146
PI, 408, 425-426
PICS, code, 1343
pixelDepth, code to access, 972
platform, 904, 909-910
Plugin object, 934
Plugin.description, 935
Plugin.filename, 935
Plugin.length, 936
Plugin.name, 937

- ul style="list-style-type: none; padding-left: 0;">
- plugins, 636, 681, 904, 910, 1109
- plugins.length, plug-ins, code to display, 682
- port, 573, 588-589, 865, 886, 892-893, 1373, 1376
- POSITIVE INFINITY, 440, 444-445
- prefix, CD:1455, CD:1468
- previous, 799, 804-805, 1117
- previousSibling, 1455, CD:1468-1469
- prevValue, CD:1756, CD:1760-1761
- primitiveType, CD:1781, CD:1785
- ProcessingInstruction object, CD:1472
- ProcessingInstruction.data, CD:1472-1473
- ProcessingInstruction.target, CD:1473
- profile, CD:1562-1563
- prompt, CD:1600-1602
- properties, 358
- protocol, 573, 589-590, 865, 886
- prototype, 16, 361, 445-446, 455, 485-486
 - Array object, 258-259
 - Boolean object, 273
 - Boolean object, 275-276
 - Cursor object, 1219
 - Date object, 307-308
 - Function object, 353
 - Number object, 440
 - Object object, 452
 - RegExp object, 467
 - String object, 500
- publicId, CD:1425, CD:1428, CD:1445-1446, CD:1470-1471
- Radio object, 938
- Radio.checked, 940
- Radio.defaultChecked, 942
- Radio.form, 944
- Radio.name, 946
- Radio.type, 949
- Radio.value, 950
- readOnly, CD:1586, CD:1595, CD:1715, CD:1720-1721
- Rect object, CD:1812-1813
- Rect.bottom, CD:1813
- Rect.left, CD:1814
- Rect.right, CD:1814
- Rect.top, CD:1815
- red, code to read, CD:1817-1818
- referrer, 637, 683, 1109, CD:1525, CD:1534
- RegExp object, 467
 - RegExp.\$, code, 471
 - RegExp.\$*, 467, 469-470
 - RegExp.\$_, 470-471
 - RegExp.\$&, 470
 - RegExp.\$', 472-473
 - RegExp.\$+, 473-474
 - RegExp.\$`, 471-472
 - RegExp.\$1,\$2,...\$9, 467, 474-475
 - RegExp.constructor, 475
 - RegExp.global, 478
 - RegExp.ignoreCase, 479
 - RegExp.input, 480
 - RegExp.lastIndex, 481
 - RegExp.lastMatch, 481
 - RegExp.lastParen, 482
 - RegExp.leftContext, 483
 - RegExp.multiline, 484
 - RegExp.prototype, 485
 - RegExp.rightContext, 486
 - RegExp.source, 487
 - rel, CD:1480, CD:1486, CD:1609, CD:1613
 - relatedNode (MutationEvent object), CD:1756
 - relatedTarget, CD:1748, CD:1753-1754
 - removeProperty() method, code, CD:1798
 - Replyto, 1355, 1364
 - Request object, 1318, 1327
 - request.agent, 1328
 - Request.Form.Count, 1322
 - request.formKey, 1328-1329
 - request.imageX, 1329-1330
 - request.imageY, 1330
 - request.ip, 1331
 - request.method, 1331
 - request.protocol, 1332
 - Request.QueryString.Count, 1323
 - Request.TotalBytes, 1326
 - requesting agents, code to view, 1327-1328
 - RESET, code to access, 740
 - Reset object, 952
 - Reset.form, 956
 - Reset.name, 958
 - Reset.type, 962
 - Reset.value, 964
 - RESIZE, code, 741
 - Response.Buffer, 1337
 - Response.CacheControl, 1337
 - Response.Charset, 1338

Response.ContentType, 1339
Response.Expires, 1340-1341
Response.ExpiresAbsolute, 1341
Response.IsClientConnected, 1342
Response.PICS, 1342-1343
Response.Status, 1343
ResultSet.prototype, 1350
rev, CD:1480, CD:1487, CD:1609,
 CD:1614
RGBColor object, CD:1816, code to
 read, CD:1816
RGBColor.blue, CD:1816
RGBColor.green, CD:1817
RGBColor.red, CD:1817
right, CD:1813-1815
rightContext, code, 467, 486
RootFolder, CD:1833, CD:1839-1840
rowIndex, CD:1703, CD:1708
rows, CD:1560-1562, CD:1688,
 CD:1698-1699, CD:1710
rowSpan, code, CD:1674,
 CD:1681-1682
rules, CD:1688, CD:1699
scheme, CD:1618, CD:1621
scope, CD:1674, CD:1682
screen object, 967, code to display,
 967-968
screen.availHeight, 968
screen.availLeft, 969
screen.availTop, 969-970
screen.availWidth, 970
screen.colorDepth, 971
screen.height, 971
screen.pixelDepth, 972
screen.width, 974
screenX, 698, 710, 1155, CD:1748,
 CD:1754
screenx (Window object), 1088
screenY, 698, 711-712, 1155, CD:1748,
 CD:1755
screeny (Window object), 1088
ScriptFullName, CD:1929, CD:1938
ScriptName, CD:1929, CD:1938-1939
ScriptTimeout, 1369, 1371
scrollbars (Window object), 1088
scrollbars.visible, code, 1158
scrolling, CD:1554, CD:1559, CD:1570,
 CD:1575
search, 573, 590-591, 865, 886, 896
sectionRowIndex, CD:1703,
 CD:1708-1709
SELECT, code, 742-743
Select object, 975
Select.form, 979
Select.length, 982
Select.name, 983
Select.options, 987
Select.options.length, 989
Select.options.selectedIndex, 989
Select.options.value, 990
Select.selectedIndex, 991
Select.type, 991
selected, 915-917, CD:1641, CD:1646
selectedIndex, 975, 990-991, CD:1659,
 CD:1667
selectorText, CD:1778-1779,
 CD:1799-1800
self, 778, 788, 1088, 1161-1162
SendMail object, 1355
SendMail.Bcc, 1357
SendMail.Body, 1357-1358
SendMail.Cc, 1358
SendMail.constructor, 1358-1359
SendMail.Errorsto, 1361
SendMail.From, 1361
SendMail.Organization, 1362
SendMail.prototype, 1362
SendMail.Replyto, 1363-1364
SendMail.Smtpserver, 1365
SendMail.Subject, 1365
SendMail.To, 1366
separator, CD:1767-1769
SerialNumber, CD:1833, CD:1840
Server object, 1369, 1373-1374
server.host, 1374
server.hostname, 1374
server.port, 1375-1376
server.protocol, 1376
Server.ScriptTimeout, 1371
Session object, 1379
Session.CodePage, 1380
Session.LCID, 1383
Session.SessionID, 1383
Session.Timeout, 1385
SessionID, 1379, 1383
setProperty() method, code, CD:1798
shape, CD:1480, CD:1487, CD:1497,
 CD:1501
ShareName, CD:1833, CD:1840-1841
shiftKey, CD:1748, CD:1755
ShortName, CD:1845, CD:1857-1858,
 CD:1891, CD:1904

- ShortPath, CD:1845, CD:1858-1859,
CD:1891, CD:1905
- siblingAbove, 822, 857
- siblingBelow, 822, 858
- size, CD:1507, CD:1546, CD:1567,
CD:1596-1597, CD:1668, CD:1906
 - HTMLBaseFontElement object,
CD:1505
 - HTMLFontElement object,
CD:1544
 - HTMLHRElement object,
CD:1565
 - HTMLInputElement object,
CD:1586
 - HTMLSelectElement object,
CD:1659
- Size, CD:1845, CD:1859-1860,
CD:1891
- Smptserver, 1355, 1365
- source, 467, 487
- Source, 1183, 1187, CD:1979,
CD:1898-1984
- SourceText, CD:1979, CD:1984
- span, CD:1684-1687
- SpecialFolders, CD:1990, CD:2003
- specified, CD:1404-1405
- SQRT1_2 (Math object), 408
- SQRT2, 408, 432
- SQRT_2, code, 431
- src, 807, 817, 822, 858-859, CD:1554,
CD:1560
- Sring (Global object), 365
- standby, CD:1624, CD:1633
- start, CD:1636-1638
- status, 1088, 1166, 1344,
CD:1977-1978, CD:1986
- Status, CD:1972, CD:1985
- statusbar (Window object), 1088
- statusbar.visible, code, 1167-1168
- StdErr, CD:1929, CD:1940,
CD:1985-1987
- StdIn, CD:1929, CD:1940-1941,
CD:1985-1988
- StdOut, CD:1929, CD:1941, CD:1985,
CD:1988-1989
- Stproc.prototype, 1392
- String.constructor, 507-508
- String.length, 514
- String.prototype, 517-518
- style, CD:1771-1772, CD:1778-1779,
CD:1800, CD:1811-1812
- Style object, 649, 662, 995
- Style.align, 997
- Style.backgroundColor, 998
- Style.backgroundImage, 999
- Style.borderBottomWidth, 1000
- Style.borderColor, 1001
- Style.borderLeftWidth, 1001
- Style.borderRightWidth, 1002
- Style.borderStyle, 1003
- Style.borderTopWidth, 1004
- Style.clear, 1006
- Style.color, 1007
- Style.display, 1007
- Style.fontFamily, 1008
- Style.fontSize, 1009
- Style.fontStyle, 1010
- Style.fontWeight, 1011
- Style.lineHeight, 1012
- Style.listStyleType, 1013
- Style.marginBottom, 1014
- Style.marginLeft, 1015
- Style.marginRight, 1016
- Style.marginTop, 1018
- Style.paddingBottom, 1018-1019
- Style.paddingLeft, 1019
- Style.paddingRight, 1020
- Style.paddingTop, 1022
- Style.textAlign, 1023
- Style.textDecoration, 1024
- Style.textIndent, 1024
- Style.textTransform, 1025
- Style.whiteSpace, 1028
- Style.width, 1028-1029
- styleSheet, CD:1772-1774
- SubFolders, CD:1891, CD:1907
- Subject, 1355, 1366
- SUBMIT, code, 743-744
- Submit object, 1030
- Submit.form, 1034-1035
- Submit.name, 1037
- Submit.type, 1041
- Submit.value, 1043
- suffixes, 899, 902
- summary, CD:1688, CD:1700
- sun (Packages object), 460
- systemId, CD:1425, CD:1429,
CD:1445-1447, CD:1456,
CD:1470-1472
- tabIndex, CD:1481, CD:1488, CD:1497,
CD:1502, CD:1513
- tagName, CD:1434, CD:1444-1445

- tags, 637, 689
- tainted objects, 20
- target, 712-713, 775-776, CD:1473,
 - CD:1488-1489, CD:1502-1505
 - Area object, 573, 592-593
 - Event object, 698, CD:1733
 - Form object, 761
 - HTMLAnchorElement object,
 - CD:1481
 - HTMLAreaElement object,
 - CD:1497
 - HTMLBaseElement object,
 - CD:1503
 - HTMLFormElement object,
 - CD:1547
 - HTMLLinkElement object,
 - CD:1609
 - Link object, 865
 - ProcessingInstruction object,
 - CD:1472
- TargetPath, CD:2005, CD:2010,
 - CD:2020-2022
- tBodies, CD:1688, CD:1700
- text, 918, CD:1511, CD:1646-1647,
 - CD:1655, CD:1658, CD:1725
 - Anchor object, 564
 - HTMLBodyElement object,
 - CD:1508
 - HTMLOptionElement object,
 - CD:1641
 - HTMLScriptElement object,
 - CD:1655
 - HTMLTitleElement object,
 - CD:1724
 - Link object, 865
 - Option object, 915
- Text object, 1046-1047
- Text.defaultValue, 1049
- Text.form, 1051
- Text.name, 1053
- Text.type, 1059
- Text.value, 1061
- textAlign, 649, 663, 688, 996, 1023
- Textarea object, 1064
- Textarea.defaultValue, 1066
- Textarea.form, 1068
- Textarea.name, 1071
- Textarea.type, 1080
- Textarea.value, 1082
- textDecoration, 649, 663, 688, 996, 1024
- textIndent, 649, 663, 688, 996, 1025
- TextStream object, CD:1912-1913
- TextStream.AtEndOfLine, CD:1913
- TextStream.AtEndOfStream, CD:1914
- TextStream.Column, CD:1915
- TextStream.Line, CD:1916
- textTransform, 650, 663, 689, 996,
 - 1025-1026
- tFoot, CD:1689, CD:1701
- tHead, CD:1689, CD:1701-1702
- Timeout, code, 1379, 1385
- timeStamp, CD:1733, CD:1740
- title, 637, 690, 1109, CD:1525, CD:1535
- To, code, 1366
- To (SendMail object), 1355
- toolbar (Window object), 1088
- toolbar.visible, code, 1169-1170
- top, 778, 790, 822, 859-860, 1088
- TotalBytes, 1318, 1326-1327
- TotalSize, CD:1833, CD:1841-1842
- type
 - alert boxes, code to display,
 - 1059-1060
 - Button object, 597
 - Checkbox object, 615
 - CSSRule object, CD:1787
 - Event object, 698, CD:1733
 - FileUpload object, 746
 - Hidden object, 793
 - HTMLAnchorElement object,
 - CD:1481
 - HTMLButtonElement object,
 - CD:1513
 - HTMLInputElement object,
 - CD:1586
 - HTMLLIElement object,
 - CD:1607
 - HTMMLLinkElement object,
 - CD:1609
 - HTMLObjectElement object,
 - CD:1624
 - HTMLOListElement object,
 - CD:1636
 - HTMLParamElement object,
 - CD:1649
 - HTMLScriptElement object,
 - CD:1655
 - HTMLSelectElement object,
 - CD:1659
 - HTMLStyleElement object,
 - CD:1670

- HTMLTextAreaElement object, CD:1715
- HTMLUListElement object, CD:1726
- MimeType object, 899
- Password object, 922
- Radio object, 938
- Reset object, 952
- Select object, 975
- Submit object, 1030
- Text object, 1046
- Textarea object, 1064
- type property
 - alert boxes, code to display, 1041-1042
 - button 's type, code to access, 609-610
 - code, 795-796, 930-931, 949, CD:1517, CD:1598
 - code to access, 631-632, 713-714, 902-903
 - code to display, 1080-1081
 - codet, CD:1489
 - displaying, code, 992
 - rule type reading, code, CD:1789
- Type, CD:1845, CD:1860, CD:1891
- UIEvent object, CD:1761
- UIEvent.detail, CD:1762
- UIEvent.view, CD:description or syntax, CD:1763
- undefined, 365, 381, 548
- UNLOAD, Event object, 699
- Unnamed, CD:1944, CD:1951
- unwatch() (MimeType object), 899
- URL, 637, 692, 1109, CD:1535-1536
- url (HTMLDocument object), CD:1525
- useMap, CD:1624, CD:1577, CD:1583-1586, CD:1599, CD:1635
- userAgent, 904, 913
- UserDomain, CD:1964, CD:1971
- UserName, CD:1964, CD:1972
- vAlign, code, CD:1674, CD:1683-1684, CD:1687, CD:1703, CD:1709-1711
- value, 612-613, 634-635, 797, 920-921, 932
 - Attr object, CD:1404
 - Button object, 598
 - Checkbox object, 615
 - FileUpload object, 746
 - Hidden object, 793
 - HTMLButtonElement object, CD:1513
 - HTMLInputElement object, CD:1586
 - HTMLLIElement object, CD:1607
 - HTMLOptionElement object, CD:1641
 - HTMLParamElement object, CD:1649
 - HTMLSelectElement object, CD:1659
 - Option object, 915
 - Password object, 922
 - Radio object, 938
 - Reset object, 952
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1064
- valueType, CD:1649, CD:1652
- version, CD:1568-1569, CD:1942
- Version (WScript object), CD:1929
- verticalAlign (Style object), 650, 663, 689
- view (UIEvent object), CD:1761
- visibility, 822, 862
- visible, code, 1123-1125, 1145-1146, 1158, 1167-1170
- vLink, CD:1508, CD:1511-1512
- vlinkColor, 637, 692-693, 1109
- VolumeName, CD:1833, CD:1842
- vpace, 808, 819, CD:1635, CD:1490, CD:1496
- watch() (MimeType object), 899
- which, 698, 718
- whiteSpace, 650, 663, 689, 996, 1028
- width, 719, 820-821, 974, 1029, CD:1497
 - Event object, 698
 - HTMLAppletElement object, CD:1490
 - HTMLHRElement object, CD:1565
 - HTMLIFrameElement object, CD:1570
 - HTMLImageElement object, CD:1577
 - HTMLObjectElement object, CD:1624
 - HTMLPreElement object, CD:1652

HTMLTableCellElement object,
CD:1674
HTMLTableColElement object,
CD:1684
HTMLTableElement object,
CD:1689
Image object, 808
Style object, 650, 663, 689, 996
window, 778, 792-793, 1088, 1173
Window object, 1087
window.closed, 1100-1101
window.crypto, 1103
window.defaultStatus, 1105-1106
window.document, 1107-1110
window.document property, 1108
window.frames, 1114
window.frames.length, 1115
window.history, 1116-1117
window.innerHeight, 1118
window.innerWidth, 1120
window.length, 1121
window.location, 1122
window.locationbar, 1122
window.locationbar.visible, 1123
window.menubar, 1123-1124
window.menubar.visible, 1124
window.name, 1127
window.offscreenBuffering, 1129
window.opener, 1137
window.outerHeight, 1139
window.outerWidth, 1140
window.pageXOffset, 1142
window.pageYOffset, 1143
window.parent, 1144
window.personalbar, 1144-1145
window.personalbar.visible, 1145
window.screenX, 1154-1155
window.screenY, 1155
window.scrollbars, 1157
window.scrollbars.visible, 1158
window.self, 1161
window.status, 1166
window.statusbar, 1167
window.statusbar.visible, 1167
window.toolbar, 1169
window.toolbar.visible, 1169
window.top, 1170
window.window, 1173
WindowStyle, code, CD:2011
WindowStyle (WshShortcut object),
CD:2005
WorkingDirectory, CD:2005, CD:2012
WScript object, CD:1929
WScript.Arguments, CD:1930
WScript.FullName, CD:1934
WScript.Interactive, CD:1935
WScript.Name, CD:1936
WScript.Path, CD:1937
WScript.ScriptFullName, CD:1938
WScript.ScriptName, CD:1938
WScript.StdErr, CD:1939
WScript.StdIn, CD:1940
WScript.StdOut, CD:1941
WScript.Version, CD:1942
WshArguments, CD:1929, CD:1944
WshArguments.Count, CD:1945
WshArguments.Length, CD:1947
WshArguments.Named, CD:1948
WshArguments.Unnamed, CD:1950
WshEnvironment, CD:1954, CD:1990
WshEnvironment.Length(), CD:1957
WshNamed object, CD:1960
WshNamed.Item, CD:1962
WshNamed.length, CD:1963
WshNetwork object, CD:1964
WshNetwork.ComputerName, CD:1966
WshNetwork.UserDomain, CD:1971
WshNetwork.UserName, CD:1971-1972
WshRemote object, CD:1972
WshRemote.Error, CD:1974-1975
WshRemote.Status, CD:1977
WshRemoteError object, CD:1979
WshRemoteError.Character, CD:1980
WshRemoteError.Description,
CD:1980-1981
WshRemoteError.Line, CD:1981
WshRemoteError.Number, CD:1982
WshRemoteError.Source, CD:1983
WshRemoteError.SourceText, CD:1984
WshScriptExec object, CD:1985
WshScriptExec.Status, CD:1985
WshScriptExec.StdErr, CD:1986
WshScriptExec.StdIn, CD:1987
WshScriptExec.StdOut, CD:1988
WshShell object, CD:1990
WshShell.CurrentDirectory, CD:1992
WshShell.Environment, CD:1993
WshShell.SpecialFolders, CD:2002
WshShortcut, CD:1990, CD:2005
WshShortcut.Arguments, CD:2005
WshShortcut.Description, CD:2006

WshShortcut.FullName, CD:2007
 WshShortcut.Hotkey, CD:2007-2008
 WshShortcut.IconLocation, CD:2008
 WshShortcut.TargetPath, CD:2010
 WshShortcut.WindowStyle,
 CD:2010-2011
 WshShortcut.WorkingDirectory,
 CD:2011
 WshSpecialFolders, CD:1990, CD:2013
 WshSpecialFolders.Item, CD:2014
 WshSpecialFolders.length, CD:2015
 WshUnnamed.Item, CD:2018
 WshUnnamed.length, CD:2019
 WshURLShortcut (WshShell object),
 CD:1990
 WshUrlShortcut object, CD:2020
 WshUrlShortcut.FullName, CD:2020
 WshUrlShortcut.TargetPath, CD:2022
 x (Anchor object), 564
 y (Anchor object), 564, 571
 zIndex, code, 822, 864

properties property, 358

property names (Button object), 71

protected keyword, 466

protocol property, 880, 893-894, 1376

accessing, code, 1332-1333
 Area object, 573, 589-590
 Link object, 865
 Location object, 886
 request object, 1327
 server object, 1373

prototype property, 16, 361, 445-446, 455, 485-486, 1228

Array object, 258-259
 Boolean object, 273-276
 Cursor object, 1219
 Date object, 307-308
 Function object, 353
 Number object, 440
 Object object, 452
 properties and methods, 1287, 1304,
 1350-1351, 1363, 1392-1393
 RegExp object, 467
 String object, 500

public keyword, 466

publicId property

code, CD:1428, CD:1446, CD:1471
 DocumentType object, CD:1425
 Entity object, CD:1445
 Notation object, CD:1470

push() method, 42, 250, 259-260

PWD parameter, 171

pwd parameter

connect() method, 1238, 1258
 DbPool() method, 1262
 DbPool object, 1256

PWS (Personal Web Server), 1175

Q

queries

databases, 1204, 1240
 DML, 1205, 1242
 SELECT, 1213, 1248-1249

QUERY STRING variable, 1325, 1387

QueryString collection, 1318, 1323

question mark (?), 87, 573, 886, 1323

Quit() method, CD:1929, CD:1937-1938

quotation marks

double (" "), 1215
 single (' '), 35

R

-r errorfile command-line parameter, 161

\r escape sequence, 36

r option (open() method), 1275, 1286

r+ option (open() method), 1275, 1286

Radio object, 938-939

Radio.blur() method, 939

Radio.checked property, 940

Radio.click() method, 941

Radio.defaultChecked property, 942

Radio.focus() method, 943

Radio.form property, 944

Radio.handleEvent() method, 944

Radio.name property, 946

Radio.onBlur event handler, 946

Radio.onClick event handler, 947

Radio.onFocus event handler, 948

Radio.type property, 949

Radio.unwatch() method, 949

Radio.value property, 950

Radio.watch() method, 951

random numbers, 1104

random() method, 409, 427-428

RangeError, 99, 344

read() method

code, 1288, CD:1917-1918
 File object, 1275
 TextStream object, CD:1912

readability (if...else statement), 64

ReadAll() method, CD:1912,

CD:1918-1919

readByte() method, 1275, 1289

reading

cookies, 138-139

formKey property, 1329

href property, CD:1772-1773

media property, CD:1774

registry entries, CD:1998

styleSheet property, CD:1774

ReadLine() method, CD:1912, CD:1919

readln() method, 1275 1290

readOnly property

code, CD:1595, CD:1720-1721

HTMLInputElement object, CD:1586

HTMLTextAreaElement object,

CD:1715

Rect object, CD:1812-1813

rect values, CD:1783

Rect.bottom property, CD:1813

Rect.left property, CD:1814

Rect.right property, CD:1814

Rect.top property, CD:1815

red property, CD:1817-1818

redirect() function, 1316, 1176

Redirect() method, 1335, 1343

redirecting user browsers, 1316

reducing

array elements, 257

repetitive code with the with statement,
75-76

reference point values (setPosition()

method), 1290

reference resources, 29-30

ReferenceError, 99, 344

references

call by value versus call by reference

(functions), 77

entity, CD:1418-1419

referrer property

code, CD:1534

Document object, 637

links, 683

HTMLDocument object, CD:1525

window.document property, 1109

RegDelete() method, CD:1990,

CD:1997-1998

RegExp (Regular Expression), 86

RegExp object, 466-468

RegExp,\$ property, 471

RegExp,\$* property, 469-470

RegExp,\$_ property, 470-471

RegExp. property, 472

RegExp.\$& property, 470

RegExp.\$' property, 472-473

RegExp.\$+ property, 473-474

RegExp.\$` property, 471-472

RegExp.\$1,\$2,...\$9 property, 467, 474-475

RegExp.compile() method, 476

RegExp.constructor property, 475

RegExp.exec() method, 477

RegExp.global property, 478

RegExp.ignoreCase property, 479

RegExp.input property, 480

RegExp.lastIndex property, 481

RegExp.lastMatch property, 481

RegExp.lastParen property, 482

RegExp.leftContext property, 483

RegExp.multiline property, 484

RegExp.prototype property, 485

RegExp.rightContext property, 486

RegExp.source property, 487

RegExp.test() method, 487-488

RegExp.toSource() method, 488

RegExp.toString() method, 489

RegExp.unwatch() method, 490

RegExp.valueOf() method, 491

RegExp.watch() method, 492

RegExpError runtime exception error, 344

registerCFunction function, 1317

registered callC() function, 1192-1193

**registering C functions in shared libraries,
1317**

registry (Windows), CD:1997-1999

RegRead() method, CD:1990, CD:1998

**Regular Expression, RegExp object,
466-468**

regular expressions, 89, 254

RegWrite() method, CD:1990, CD:1999

rel property

code, CD:1486, CD:1613

HTMLAnchorElement object, CD:1480

HTMLLinkElement object, CD:1609

**relatedNode property (MutationEvent
object), CD:1756**

**relatedTarget property, CD:1748,
CD:1753-1754**

relationships, parent/child, 139-141

release() method, 1199, 1210-1211

releaseEvents() method

code, 684-685, 852-853

Document object, 637

- Layer object, 822
- Window object, 1086
 - window.document property, 1108
- releasing events, 683-684**
- reload() method, 886, 894-895**
- REMOTE ADDR variable, 1325, 1387**
- REMOTE HOST variable, 1326, 138**
- remote scripts, 187-189**
 - creating, CD:1953-1954
 - starting, CD:1976
- REMOTE USER variable, 1326, 1387**
- REMOVAL constant (MutationEvent object), CD:1756**
- Remove control option (Server Side JavaScript Application Manager), 166**
- Remove() method**
 - code, 1382, CD:1666-1667
 - Dictionary object, CD:1825, CD:1831-1832
 - environment variables, code to delete, CD:1959
 - WshEnvironment object, CD:1954
- remove() method (HTMLSelectElement object), CD:1659**
- RemoveAll() method, 1382-1383, CD:1825, CD:1832**
- removeAttribute() method, CD:1435, CD:1440**
- removeAttributeNode() method, CD:1435, CD:1441**
- removeAttributeNS() method, CD:1435, CD:1441**
- removeChild() method, CD:1456, CD:1469**
- removeEventListener() method, CD:1744, CD:1747**
- removeMember() method, 397, 404**
- removeNamedItem() method, CD:1451**
- removeNamedItemNS() method, CD:1451-1452**
- RemoveNetworkDrive() method, CD:1964, CD:1969**
- RemovePrinterConnection() method, CD:1964, CD:1970**
- removeProperty() method, CD:1793, CD:1798**
- removing**
 - alert() method, 106
 - array elements, 257, 261-262
 - button focus, 599-600
 - check box focus, 617-618
 - focus from text areas, 1065-1066
 - items in dictionaries, CD:1831-1832
 - select box focus, 977
 - Submit button focus, 1032
 - text box focus, 1048
 - window focus, 1092-1093
- repetitive code, 75-76**
- replace() method, 499, 886, 895**
- replace(reqExpObj,str) method, 88**
- replaceChild() method, CD:1456, CD:1470**
- replaceData() method (CharacterData object), CD:1407**
- replicateData() method, CD:1411-1412**
- Replyto property, 1355, 1364**
- REQUEST METHOD variable, 132, 1387**
- Request object, 168, 176, 1318, 1327-1328**
- request.agent property, 1328**
- Request.BinaryRead() method, 1318-1319**
- Request.ClientCertificate collection, 1319**
- Request.Cookies collection, 1321**
- Request.Form collection, 177, 1321**
- Request.Form.Count property, 1322**
- request.formKey property, 1328-1329**
- request.imageX property, 1329-1330**
- request.imageY property, 1330**
- request.ip property, 1331**
- request.method property, 1331**
- request.protocol property, 1332**
- Request.QueryString collection, 1323**
- Request.QueryString.Count property, 1323**
- Request.ReadBinary() method, 1322**
- Request.ServerVariables collection, 177-178, 1324**
- Request.TotalBytes property, 1326**
- request.unwatch() method, 1333**
- request.watch() method, 1334**
- reserved words. See keywords**
- Reset button, 76**
- Reset event, 91**
- RESET event (Event object), 699**
- Reset object, 952-953**
- RESET property, 740**
- reset() method, 761, 773, CD:1547, CD:1552-1553**
- Reset.blur() method, 953**
- Reset.click() method, 954**
- Reset.focus() method, 955**
- Reset.form property, 956**
- Reset.handleEvent() method, 957**
- Reset.name property, 958**
- Reset.onBlur event handler, 959**
- Reset.onClicK event handler, 960**

Reset.onFocus event handler, 961

Reset.type property, 962

Reset.unwatch() method, 962-963

Reset.value property, 964

Reset.watch() method, 965

ResetFields() funtion, 76

resetting check boxes, 621-622

resizable parameter (Window object), 1135

Resize event, 91

RESIZE event (Event object), 699

RESIZE property, 741

resizeBy() method, 822, 854, 1086, 1150-1151

resizeTo() method, 822, 855, 1086, 1152

resources

general information, 28-29

newsgroups, 30

references, 29-30

Response object, 177, 1334-1335

Response object (JScript), 168

Response.AddHeader() method, 1335

Response.AppendToLog() method, 1336

Response.BinaryWrite() method, 1336

Response.Buffer property, 1337

Response.CacheControl property, 1337

Response.Charset property, 1338

Response.Clear() method, 1338

Response.ContentType property, 1339

Response.Cookies collection, 1339-1340

Response.End() method, 1340

Response.Expires property, 1340-1341

Response.ExpiresAbsolute property, 1341

Response.Flush() method, 1341-1342

Response.IsClientConnected property, 1342

Response.PICS property, 1342-1343

Response.Redirect() method, 1343

Response.Status property, 1343

Response.Write() method, 1344

Restart control option (Server Side JavaScript Application Manager), 165

Resultset object, 1344-1346

resultSet() method, 1388, 1394

Resultset.close() method, 1346

Resultset.columnName() method, 1347

Resultset.columns() method, 1348

Resultset.next() method, 1349

Resultset.prototype property, 1350

Resultset.unwatch() method, 1351

Resultset.watch() method, 1353

resultsets, closing, 1346-1347

retrieving

arguments, CD:1942-1943

array of items, CD:1829

array of keys, CD:1830-1831

attributes, CD:1846-1847

drives, CD:1844-1845

file creation dates, CD:1848-1849

free space, CD:1837-1838

network drive mappings, CD:1967

script mode, CD:1936

Submit button names, 1037-1038

WSH directory, CD:1937

WSH name, CD:1936

return keyword, 493

return statements, 79, 493

return values

Array object, 249

Boolean object, 273

concat() method, 251

dbInstance (toString() method), 1215, 1252, 1269

dbName (toString() method), 1215, 1252, 1269

dbType (toString() method), 1215, 1252, 1269

event handlers, 94-95

join() method, 255

pop() method, 257

push() method, 259

shift() method (Array object), 261

slice() method (Array object), 262

splice() method (Array object), 266

toString() method, 268, 277, 1215, 1252

uid (toString() method), 1215, 1252, 1269

unshift() method (Array object), 269

UTC() method (Date object), 326

returning values, 60, 79-80

returnValue() method, 1388, 1395

rev property, CD:1480, CD:1487, CD:1609, CD:1614

reverse() method (Array object), 42, 250, 260-261

reversing array elements, 260-261

RGBColor object, CD:1815-1816

RGBColor.blue property, CD:1816

RGBColor.green property, CD:1817

RGBColor.red property, CD:1817

right property, CD:1813-1815

rightContext property, 467, 486

rollbackTransaction() method, 1199, 1211-1212, 1234, 1247
rolling back transactions, 1247
root folder, CD:1839-1840
root key abbreviations, CD:1997-1999
RootFolder property, CD:1833, CD:1839-1840
round() method, 409, 428-429
routeEvent() method, 637, 822, 855-856, 686-687
rowIndex property, CD:1703, CD:1708
rows
 iterating through, 1227-1228
 updating, 1231-1232
rows property
 code, CD:1562, CD:1698-1699, CD:1713-1714, CD:1721
 HTMLFrameSetElement object, CD:1560
 HTMLTableElement object, CD:1688
 HTMLTableSectionElement object, CD:1710
 HTMLTextAreaElement object, CD:1715
rowSpan property, CD:1674, CD:1681-1682
rules
 CSSRule object, CD:1787
 deleteRule() method, CD:1776, CD:1802
 insertRule() method, CD:1777, CD:1803
 property, CD:1688, CD:1699
 type conversions, 57
Run control option (Server Side JavaScript Application Manager), 165
Run() method, CD:1990, CD:2000
Run option (scripts), 185
running
 DML queries, 1242
 scripts from Run option, 185
runtime (JScript), 15
runtime errors, 99-100, 344

S

"s" fruits, finding in dictionary, CD:1827
S subfield (ClientCertificate collection), 1320
Same Origin Policy, 19

Save() method, CD:2005, CD:2009-2010, CD:2020-2022
scheme property, CD:1618, CD:1621
scope (variables), 38-39
scope property, CD:167, CD:1682
screen object, 967-968
screen.availHeight property, 968
screen.availLeft property, 969
screen.availTop property, 969-970
screen.availWidth property, 970
screen.colorDepth property, 971
screen.height property, 971
screen.pixelDepth property, 972
screen.unwatch() method, 972
screen.watch() method, 973
screen.width property, 974
screenX parameter (Window object), 1135
screenX property
 code, 710, 1155, CD:1754
 Event object, 698
 MouseEvent object, CD:1748
 Window, 1088
screenY parameter (Window object), 1135
screenY property, 711-712
 code, 1155, CD:1755
 Event object, 698
 MouseEvent object, CD:1748
 Window, 1088
Script Host (Windows), WSH (Windows Script Host), 7, 16, 27-28, 179-189, CD:1936-1937
script management, 165-167
script mode, CD:1936
SCRIPT NAME variable, 1326, 138
<script> tag, 22, 32-33, 111-112, 159, 180
ScriptEngine() function, 494
ScriptEngineBuildVersion() function, 494-495
ScriptEngineMajorVersion() function, 495
ScriptEngineMinorVersion() function, 495-496
ScriptFullName property, CD:1929, CD:1938
scripting, 108, 179
 client-side
 cookies, 136-139
 dynamic documents, 145-154
 form processing, 124-136
 Web browsers, 108-123
 windows, 139-145
 XUL (XML-based User Interface Language), 154-156

ECMAScript, 7, 10, 193
engines, 7, 112-114, CD:1823
remote, 187-189
Windows, 1179

Scripting Engine (Microsoft), CD:1823
Scripting Runtime Library (Microsoft), CD:1824

Scripting Technologies (Microsoft), CD:1824

Scripting.Signer object, CD:1927-1928

ScriptingContext object, 1354

ScriptName property, CD:1929,

CD:1938-1939

scripts

compiling, 160-161
debugging, 103-106
removing, 102-106
executing from command lines, 185
Hello World, 185-186
lines, ending, 32
managing, 165-167
MSSD, 102-103
remote, CD:1953-1954
running from Run option, 185
signed, 21-23
starting, CD:1976
usage information, CD:1950
WSH (Windows Script Host), 184-187

ScriptTimeout property, 1369-1371

scroll() method, 1086, 1156-1157

scrollbars parameter (Window object), 1135

scrollbars property (Window), 1088

scrollbars.visible property, 1158, 1171

scrollBy() method, 1159-1160, 1086

scrolling property, CD:1554, CD:1559, CD:1570, CD:1575

scrollTo() method, 1086, 1160-1161

scrrun.dll file, CD:1824

search property

Area object, 573, 590-591
code, 880-881, 896
Link object, 865
Location object, 886

search() method (String object), 499

search(reqExpObj) method, 88

seconds argument, 286, 326

sectionRowIndex property, CD:1703, CD:1708-1709

secure attribute (cookies), 137

security, 18-23

select boxes, 977-979, 982-984, 988-989

Select event, 91

SELECT event (Event object), 699

Select object, 974-976

SELECT property, 742-743

SELECT queries, 1213, 1248-1249

<select> tag, 1299

select() method

code, 756-757, 930, CD:1596, CD:1722

FileUpload object, 746

HTMLInputElement object, CD:1586

HTMLTextAreaElement object,
CD:1715

Password object, 923

text areas, 1079-1080

text boxes, 1059

Text object, 1046

Textarea object, 1063

Select.blur() method, 976-977

Select.focus() method, 978

Select.form property, 979

Select.handleEvent() method, 980

Select.length property, 982

Select.name property, 983

Select.onBlur event handler, 984

Select.onChange event handler, 985

Select.onFocus event handler, 986

Select.options property, 987

Select.options.length property, 989

Select.options.selectedIndex property, 989

Select.options.value property, 990

Select.selectedIndex property, 991

Select.type property, 991

Select.unwatch() method, 992

Select.watch() method, 993

selected property, 915-917, CD:1641, CD:1646

selected text, displaying, 659-660

selectedIndex property, 975, 990-991, CD:1659, CD:1667

selecting

array subsections, 262-263

text box text, 1059

selectorText property, CD:1779-1800

self property, 778, 1088, 1161-1162

semicolon (;), 31-32

send() method, 1355, 1364-1365

sending

e-mail, 1360-1361

events, CD:1799

keystrokes to Notepad, CD:2002

strings, CD:1940-1941

- SendKeys() method, CD:1990, CD:2002
- SendMail object, 172-173, 1354-1357
- SendMail.Bcc property, 1357
- SendMail.Body property, 1357-1358
- SendMail.Cc property, 1358
- SendMail.constructor property, 1358-1359
- SendMail.errorCode() method, 1359
- SendMail.errorMessage() method, 1360-1361
- SendMail.Errorsto property, 1361
- SendMail.From property, 1361
- SendMail.Organization property, 1362
- SendMail.prototype property, 1362
- SendMail.Replyto property, 1363-1364
- SendMail.send() method, 1364
- SendMail.Smtpserver property, 1365
- SendMail.Subject property, 1365
- SendMail.To property, 1366
- SendMail.unwatch() method, 1366
- SendMail.watch() method, 1367
- separating code, 32
- separator property, CD:1767-1769
- SerialNumber field (ClientCertificate collection), 1320
- SerialNumber property, CD:1833, CD:1840
- SERVER NAME variable, 1326, 1387
- Server object, 168, 1368-1374
- SERVER PORT SECURE variable (ServerVariables collection), 1326
- SERVER PORT variable, 1326, 1387
- SERVER PROTOCOL variable, 1326, 1387
- Server Side JavaScript Application Manager, 161-167
- SERVER SOFTWARE variable (ServerVariables collection), 1326
- SERVER URL variable (ssjs_getCGIVariable() function), 1387
- <server> tag, 158-159
- server-side
 - versus client-side, 158-159
 - environments, 158
 - JavaScript, 13, 159-162, 169-178
 - JSript, 168
 - versatility, 24-25
- Server.CreateObject() method, 1369
- Server.Execute() method, 1369-1370
- Server.GetLastError() method, 1370
- server.host property, 1374
- server.hostname property, 1374
- Server.HTMLEncode() method, 1370
- server.lock() method, 1375
- Server.MapPath() method, 1371
- server.port property, 1375-1376
- server.protocol property, 1376
- Server.ScriptTimeout property, 1371
- Server.Transfer() method, 1372
- server.unlock() method, 1376
- server.unwatch() method, 1377
- Server.URLEncode() method, 1372
- server.watch() method, 1378
- serverName argument (ActiveXObject object), 242
- servers
 - ASP environment, 168-169
 - COM (Component Object Model), CD:1824
 - Enterprise Server, 170-176
 - IIS (Internet Information Server), 157, 176-178, 1175
 - iPlanet Enterprise Web site, 157
 - NES (Netscape Enterprise Server), 563
 - Netscape server-side JavaScript support, 160
 - PWS (Personal Web Server), 1175
 - variables, 177-178
- ServerVariables collection, 1318, 1324-1326
- Session object, 168t, 1378-1379
- Session OnEnd event (Session object), 1379
- Session OnStart event (Session object), 1379
- Session.Abandon() method, 1379
- Session.CodePage property, 1380
- Session.Contents collection, 1381
- Session.Contents.Remove() method, 1381
- Session.Contents.RemoveAll() method, 1382
- Session.LCID property, 1383
- Session.SessionID property, 1383
- Session.Session_OnEnd event, 1384
- Session.Session_OnStart event, 1384
- Session.StaticObjects collection, 1385
- Session.Timeout property, 1385
- SessionID property, 1379, 1383
- Session_OnEnd event, 1384
- SetAbort() method, 1308, 1311
- setAttribute() method, CD:1435, CD:1442
- setAttributeNode() method, CD:1435, CD:1442-1443
- setAttributeNodeNS() method, CD:1435, CD:1443

- setAttributeNS() method, CD:1435, CD:1444
- setColor() function, 104
- SetComplete() method, 1309-1312
- setDate() method (Date object), 287, 308-309
- SetDefaultPrinter() method, CD:1964, CD:1970-1971
- setFloatValue() method, CD:1781, CD:1785-1786
- setFullYear() method (Date object), 287, 309-310
- setHotKeys() method, 1086, 1162
- setHours() method (Date object), 287, 310
- setInterval() method, 95-96, 779 789, 1086, 1163
- setMember() method, 397, 405
- setMilliseconds() method (Date object), 287, 310-311
- setMinutes() method (Date object), 287, 311-312
- setMonth() method (Date object), 287, 312-313
- setNamedItem() method, CD:1452
- setNamedItemNS() method, CD:1453
- setPosition() method, 1275, 1291-1290
- setProperty() method, CD:1793, CD:1798
- setResizable() method, 1086, 1164
- setSeconds() method (Date object), 287, 313-314
- setSlot() method, 397, 406
- setStringValue() method, CD:1781, CD:1786
- setTime() method (Date object), 287, 314
- setTimeout() method, 96, 779, 789-790, 1086, 1164-1165
- setting
 - alink property, 638
 - Submit button focus, 1034
- setType() function, 104
- setUTCDate() method (Date object), 287, 314-315
- setUTCFullYear() method (Date object), 287, 315-316
- setUTCHours() method (Date object), 287, 316-317
- setUTCMilliseconds() method (Date object), 287, 317
- setUTCMinutes() method (Date object), 287, 318
- setUTCMonth() method (Date object), 287, 318-319
- setUTCSeconds() method (Date object), 288, 319-320
- setYear() method (Date object), 288, 320-321
- setZOptions() method, 1086, 1166
- 17-inch monitors, global variables, 39
- shape property, CD:1480, CD:1487, CD:1497, CD:1501
- shared libraries, 1317
- ShareName property, CD:1833, CD:1840-CD:1841
- shift left (<<) bitwise operator, 61
- shift left assignment operator (<<=), 230-231
- shift left operator (<<), 229-230
- shift() method, 42, 250, 261-262
- shift right with sign assignment operator (>>=), 238-239
- shift right with sign operator (>>), 237-238
- shift right with sign(>>) bitwise operator, 61
- shift right zero fill (>>>) bitwise operator, 61
- shift right zero fill assignment operator (>>>=), 240-241
- shift right zero fill operator (>>>), 239-240
- shiftKey property, CD:1748, CD:1755
- short keyword, 496
- short names of files, CD:1858
- short paths of files, CD:1858-1859
- shortcuts, creating, 183
- ShortName property, CD:1845, CD:1857-1858, CD:1891, CD:1904
- ShortPath property (File object), CD:1845, CD:1858-1859, CD:1891, CD:1905
- showCar function, 130
- showColor function, 130
- ShowUsage() method, CD:1944, CD:1949-1950
- siblingAbove property, 822, 857
- siblingBelow property, 822, 858
- signed scripts, 21-23
- simulating click events, 600-601
- sin() method, 409, 429-430
- single quotation marks (' '), 35
- single-line comments (//), 216-217
- size property
 - code, CD:1507, CD:1546, CD:1567, CD:1596-1597, CD:1668, CD:1906
 - File object, CD:1845, CD:1859-1860
 - Folder object, CD:1891

- HTMLBaseFontElement object, CD:1505
- HTMLFontElement object, CD:1544
- HTMLHRElement object, CD:1565
- HTMLInputElement object, CD:1586
- HTMLSelectElement object, CD:1659
- sizes of files, CD:1859-1860, CD:1882
- sizes of fonts, 1009
- Skip() method, CD:1912, CD:1920
- SkipLine() method, CD:1912, CD:1921
- slashes
 - / (forward), 86, 573, 587, CD:2016
 - \ (backslash), 35, 87, 160
- Sleep() method, CD:1929, CD:1939
- slice() method, 42, 250, 262-263, 499
- small() method (String object), 499
- Smtpsrv property, 1355, 1365
- sort() method, 42, 250, 263-265
- sorting array elements, 264-265
- source parameter, CD:1867-1868
- source property
 - ASPErr object, 1183
 - code, 487, 1187, CD:1983-1984
 - RegExp object, 467
 - WshRemoteError object, CD:1979
- sources
 - Array object, 267-268
 - Date object, 323
- SourceText property, CD:1979, CD:1984
- span property, CD:1684-1687
- tag, 147
- special characters (strings), 35-36
- special folder types, CD:2012-2013
- special values of numbers, 35
- SpecialFolders property, CD:1990, CD:2003
- specified property (Attr object), CD:1404
- splice() method (Array object), 42, 250, 265-266
- split() method, 88, 499
- splitText() method, CD:1474-1475
- SQLTable() method, 1199, 1213, 1234, 1248-1249
- sqr() method, 409, 430-431
- SQRT1_2 property, 408 431
- SQRT2 property, 408
- SQRT12 property, 432
- square brackets [] operator, 40
- src property
 - code, 817, 858-859, CD:1560, CD:1576, CD:1583, CD:1597
 - HTMLFrameElement object, CD:1554
 - HTMLIFrameElement object, CD:1570
 - HTMLImageElement object, CD:1577
 - HTMLInputElement object, CD:1586
 - HTMLScriptElement object, CD:1655
 - Image object, 807
 - Layer object, 822
- String property (Global object), 365
- ssjs_generateClientID() function, 1385-1386
- ssjs_getCGIVariable() function, 1386-1387
- ssjs_getClientID() function, 1387-1388
- standard error output, CD:1940
- standard output, CD:1941
- Standard Server-Side JavaScript error, 167
- standby property, CD:1624, CD:1633
- start argument (Array object), 262, 266
- Start control option (Server Side JavaScript Application Manager), 165
- Start event, CD:1972, CD:1976-1977
- start property, CD:1636-1638
- starting
 - MSSD (Microsoft Script Debugger), 102-103
 - remote scripts, CD:1976
 - scripts, CD:1976
 - transactions, 1201-1203, 1236-1237
- statements
 - @cc_on, 200
 - @if, 200-201
 - break, 75
 - conditional, 62-64
 - continue, 73-75, 283-285
 - else, 334, 389, 490
 - else...if, 64
 - if, 59, 62-63, 381-382
 - if statement, 218-219
 - if...else, 63-64, 382-383
 - labeling, 72-74
 - return, 79, 493
 - switch, 65, 537-539
 - throw, 97
 - try...catch, 97-99
 - try...catch...finally, 544-546
 - while, 560-561
 - with, 75-76, 561-562
- states of instances (vehicle objects), 8-9
- static keyword, 496-497
- StaticObjects collection, 1177, 1379, 1385
- status property
 - code, 1166, 1344, CD:1977-1978, CD:1986
 - Window, 1088

- WshRemote object, CD:1972
- WshScriptExec object, CD:1985
- statusbar property (Window), 1088
- statusbar.visible property, 1167-1168
- StdErr property, CD:1929, CD:1940, CD:1985-1987
- StdIn property, CD:1929, CD:1940-1941, CD:1985-1988
- StdOut property, CD:1929, CD:1941, CD:1985, CD:1988-1989
- stop argument (Array object), 262
- Stop control option (Server Side JavaScript Application Manager), 165
- stop() method, 1087, 1168-1169
- stopping loops, 337-338
- stopPropagation() method, CD:1733, CD:1739
- stored procedures, 1214-1215, 1250-1251, 1268-1269
- storedProc() method, 1200, 1214-1215, 1234, 1250
- storedProcArgs() method, 1234, 1251, 1257, 1268-1269
- Stproc object, 1388-1389
- Stproc.close() method, 1389
- Stproc.outParamCount() method, 1390
- Stproc.outParameters() method, 1391
- Stproc.prototype property, 1392
- Stproc.resultSet() method, 1393
- Stproc.returnValue() method, 1395
- Stproc.unwatch() method, 1395
- Stproc.watch() method, 1397
- (str) method, 89
- streams (document output), 651-652
- strike() method (String object), 499
- string argument, 255, 286, 597
- string concatenations, 53
- string data, CD:1408
- String object, 88, 497-500
- string operator, 49-50
- string values, CD:1786
- String() function, 497
- String() method, 379
- String.anchor() method, 501, 553
- String.big() method, 502
- String.blink() method, 503
- String.bold() method, 503-504
- String.charAt() method, 504-505
- String.charCodeAtAt() method, 505-506
- String.concat() method, 507
- String.constructor property, 507-508
- String.fixed() method, 508
- String.fontcolor() method, 509
- String.fontSize() method, 509-510
- String.fromCharCode() method, 510-511
- String.indexOf() method, 511-512
- String.italics() method, 512-513
- String.lastIndexOf() method, 513-514
- String.length property, 514
- String.link() method, 515
- String.localeCompare() method, 515-516
- String.match() method, 516-517
- String.prototype property, 517-518
- String.replace() method, 519
- String.search() method, 520-521
- String.slice() method, 521
- String.small() method, 522
- String.split() method, 522-523
- String.strike() method, 524-525
- String.sub() method, 525
- String.substr() method, 526
- String.substring() method, 527-528
- String.sup() method, 528
- String.toLocaleLowerCase() method, 528-529
- String.toLocaleUpperCase() method, 529-530
- String.toLowerCase() method, 530-531
- String.toSource() method, 531
- String.toString() method, 532
- String.toUpperCase() method, 533
- String.unwatch() method, 533-534
- String.valueOf() method, 534-535
- String.watch() method, 535-536
- strings
 - array indexes, 40-41
 - concatenating, 50
 - converting bytes read, 1277-1278
 - double/single quotes, 35
 - escape sequences and special characters, 35-36
 - to files, CD:1855-1856
 - read, code to convert into bytes, 1291-1292
 - sending to standard error output, CD:1940
 - sending to standard output, CD:1941
 - special characters, 35-36
 - writing to files, 1295-1298
- stringToByte() method (File object), 1275, 1291-1292
- Style object, 649-650, 662-663, 994-996

style property

- manipulating, CD:1771-CD:1772
- reading, CD:1811-1812
- setting, CD:1778-CD:1779
- writing, CD:1800

style sheets, 149, 650, 663-664, 689,
CD:1810-1811

Style.align property, 997

Style.backgroundColor property, 998

Style.backgroundImage property, 999

Style.borderBottomWidth property, 1000

Style.borderColor property, 1001

Style.borderLeftWidth property, 1001

Style.borderRightWidth property, 1002

Style.borderStyle property, 1003

Style.borderTopWidth property, 1004

Style.borderWidths() method, 1004-1005

Style.clear property, 1006

Style.color property, 1007

Style.display property, 1007

Style.fontFamily property, 1008

Style.fontSize property, 1009

Style.fontStyle property, 1010

Style.fontWeight property, 1011

Style.lineHeight property, 1012

Style.listStyleType property, 1013

Style.marginBottom property, 1014

Style.marginLeft property, 1015

Style.marginRight property, 1016

Style.margins() method, 1016

Style.marginTop property, 1018

Style.paddingBottom property, 1018-1019

Style.paddingLeft property, 1019

Style.paddingRight property, 1020

Style.paddings() method, 1021

Style.paddingTop property, 1022

Style.textAlign property, 1023

Style.textDecoration property, 1024

Style.textIndent property, 1024

Style.textTransform property, 1025

Style.unwatch() method, 1026

Style.watch() method, 1027

Style.whiteSpace property, 1028

Style.width property, 1028-1029

styles, CD:1809-1810, CD:1818-1819,
CD:2000, CD:2011

styleSheet property, CD:1772-1774

sub() method (String object), 499

subfields, 1319-1320

SubFolders property, CD:1891, CD:1907

Subject field (ClientCertificate collection),
1320

Subject property, 1355, 1366

Submit button, 1032-1034

Submit event, 91, 94

SUBMIT event (Event object), 699

Submit object, 1029-1030, 1043-1044

SUBMIT property, 743-744

submit() method, 761, 774, CD:1547,
CD:1553

Submit.blur() method, 1031-1032

Submit.click() method, 1032-1033

Submit.focus() method, 1034

Submit.form property, 1034-1035

Submit.handleEvent() method, 1036

Submit.name property, 1037

Submit.onBlur event handler, 1038

Submit.onClick event handler, 1039

Submit.onFocus event handler, 1040

Submit.type property, 1041

Submit.unwatch() method, 1042

Submit.value property, 1043

Submit.watch() method, 1044

substr() method (String object), 499

substring() method (String object), 499

substringData() method, CD:1407-1409

substrings, CD:1409

subtraction assignment operator (-=),
232-233

subtraction operator (-), 46-47, 202-203

suffixes property, 899, 902

summary property, CD:1688, CD:1700

sun object, 536-537

sun package, 463-464

sun property (Packages object), 460

sup() method (String object), 499

super keyword, 537

support

- Netscape server-side JavaScript, 160
- resources, 28-30

- Web browsers, 108-115

switch statement, 64-65, 537-539

switches, CD:1936

synchronized keyword, 539

SYNTAX_ERR constant (DOMException
object), CD:1430

SyntaxError, 99, 344

system directory path, CD:1955-1957

System.out.println() method, 121

SYSTEMDRIVE (Windows environment
variable), CD:1955

systemId property

- DocumentType object, CD:1425
- Entity object, CD:1445
- Notation object, CD:1470
- reading, CD:1429, CD:1447, CD:1456, CD:1471-1472

SYSTEMROOT (Windows environment variable), CD:1955

T

\t escape sequence, 36

T subfield (ClientCertificate collection), 1320

tagIndex property

- code, CD:1488, CD:1516-1517, CD:1598, CD:1633-1634
- HTMLAnchorElement object, CD:1481
- HTMLAreaElement object, CD:1497
- HTMLButtonElement object, CD:1513
- HTMLInputElement object, CD:1586
- HTMLObjectElement object, CD:1624
- HTMLSelectElement object, CD:1659

<table> tag, 1199

tables, 66-67, 1223-1224, 1347-1348

tagIndex property, CD:1502

tagName property, CD:1434, CD:1444-1445

tags. See HTML, tags

tags() method, 639-641

tags property, 637, 689

taint() method, 1045

taintEnabled() method, 21, 904, 912-913

tainting data (security), 20

tan() method, 409, 433

target property

- code, 881-882, CD:1488-1489, CD:1502-1505
- Event object, 698, CD:1733
- Form object, 761
- HTMLAnchorElement object, CD:1481
- HTMLAreaElement object, CD:1497
- HTMLBaseElement object, CD:1503
- HTMLFormElement object, CD:1547
- HTMLLinkElement object, CD:1609
- Link object, 865
- ProcessingInstruction object, CD:1472
- modifying, 592-593
- reading, CD:1473
- results, 775-776

TargetPath property, CD:2005, CD:2010, CD:2020-2022

tBodies property, CD:1688, CD:1700

techniques, programming, 85-106

TEMP (Windows environment variable), CD:1955

Terminate() method, CD:1972, CD:1978, CD:1985, CD:1989

test() method, 467, 488

test(str) method, 89

testing

- ASP environment, 169
- connections, 1203-1204, 1239-1240
- database connections, 1260
- pattern matches, 88-89

text

- applets, 122-123
- areas, 1066-1072, 1079-1083
- boxes, 1048-1051, 1054, 1059-1062
- buttons, 612-613
- displaying, 694-696
- entering, CD:1940-1941
- find() method, 1112
- Hello World, 187
- in applets, 123
- in text boxes, 1059
- italics, 652
- selected, 659-660
- splitText() method, CD:1474-1475

text data length, CD:1408-1409

TEXT NODE constant (Node object), CD:1454

Text object, 1046-1047, CD:1474

text property

- Anchor object, 564
- code, 882-883, 918, CD:1511, CD:1646-1647
- HTMLBodyElement object, CD:1508
- HTMLOptionElement object, CD:1641
- HTMLScriptElement object, CD:1655
- HTMLTitleElement object, CD:1724
- Link object, 865
- Option object, 915

Text.blur() method, 1048

Text.defaultValue property, 1049

Text.focus() method, 1050

Text.form property, 1051

Text.handleEvent() method, 1052

Text.name property, 1053

Text.onBlur event handler, 1054

Text.onChange event handler, 1055

Text.onFocus event handler, 1056

Text.onSelect event handler, 1057

- Text.select() method, 1058
- Text.splitText() method, CD:1474
- Text.type property, 1059
- Text.unwatch() method, 1060
- Text.value property, 1061
- Text.watch() method, 1062
- textAlign property, 649, 663, 688, 996, 1023
- Textarea object, 1063-1065
- Textarea.blur() method, 1065
- Textarea.defaultValue property, 1066
- Textarea.focus() method, 1067
- Textarea.form property, 1068
- Textarea.handleEvent() method, 1070
- Textarea.name property, 1071
- Textarea.onBlur event handler, 1072
- Textarea.onChange event handler, 1073
- Textarea.onFocus event handler, 1074
- Textarea.onKeyDown event handler, 1075
- Textarea.onKeyPress event handler, 1076
- Textarea.onKeyUp event handler, 1077
- Textarea.onSelect event handler, 1078
- Textarea.select() method, 1079
- Textarea.type property, 1080
- Textarea.unwatch() method, 1081
- Textarea.value property, 1082
- Textarea.watch() method, 1083
- textboxes, 1039-1040
- textDecoration property, 649, 663, 688, 996, 1024
- textIndent property, 649, 663, 688, 996, 1025
- TextStream object, CD:1912-1913
- TextStream.AtEndOfLine property, CD:1913
- TextStream.AtEndOfStream property, CD:1914
- TextStream.Close() method, CD:1915
- TextStream.Column property, CD:1915
- TextStream.Line property, CD:1916
- TextStream.Read() method, CD:1917
- TextStream.ReadAll() method, CD:1918
- TextStream.ReadLine() method, CD:1919
- TextStream.Skip() method, CD:1919-1920
- TextStream.SkipLine() method, CD:1920
- TextStream.Write() method, CD:1921
- TextStream.WriteBlankLines() method, CD:1922
- TextStream.WriteLine() method, CD:1923
- textTransform property, 650, 663, 689, 996, 1025-1026
- tFoot property, CD:1689, CD:1701
- tHead property, CD:1689, CD:1701-1702
- this keyword, 539-541
- throw keyword, 541-543
- throw statement, 97
- throws keyword, 543
- time, 321-325, 1098-1099
- timeout parameter, 171
- Timeout property, 1379, 1385
- timers, 95-96
- times, modularizing, 26
- timestamp property, CD:1733, CD:1740
- title property
 - accessing, 690
 - code, CD:1535
 - Document object, 637
 - HTMLDocument object, CD:1525
 - HTMLElement object, CD:1539
 - setting, CD:1542
 - window.document property, 1109
- titlebar parameter (Window object), 1135
- TMP (Windows environment variable), CD:1955
- To property, 1355, 1366
- toArray() method (VBArray object), 551, CD:1829
- toExponential() method, 440, 446
- toFixed() method, 440, 447
- toGMTString() method (Date object), 288, 321
- toLocaleLowerCase() method (String object), 499
- toLocaleString() method, 288, 322, 440, 447-448, 500
- toLowerCase() method (String object), 500
- toolbar parameter (Window object), 1135
- toolbar property (Window), 1088
- toolbar.visible property, 1169-1170
- top property, 1088
 - code, 790, 859-860, 1170-1171
 - Frame object, 778
 - Layer object, 822
 - reading, CD:1815
 - Rect object, CD:1813
- toPrecision() method, 440, 448
- toSource() method
 - Array object, 250, 267-268
 - Boolean object, 276-277
 - code, 362, 434, 449, 455-456, 488-489
 - Date object, 288, 322-323
 - Function object, 353

- Math object, 409
- Number object, 440
- Object object, 453
- String object, 500
- toString() method**
 - arrays, 42, 250, 268-269
 - Boolean object, 274, 277-278
 - code, 363, 392, 406, 434-435, 449
 - Connection object, 1200
 - Date object, 288, 323-234
 - database object, 1234
 - dbInstance return value, 1215, 1252, 1269
 - dbName return value, 1215, 1252, 1269
 - DbPool object, 1257
 - dbType return value, 1215, 1252, 1269
 - Function object, 353
 - JSObject object, 397
 - Math object, 409
 - Number object, 440
 - Object object, 453
 - RegExp object, 467
 - return values, 1215, 1252
 - String object, 500
 - uid return value, 1215, 1252, 1269
- toString() object, 390**
- TotalBytes property, 1318, 1326-1327**
- TotalSize property, CD:1833, CD:1841-1842**
- toUpperCase() method (String object), 500**
- toUTCString() method (Date object), 288, 324-325**
- Trace Information window, 1273**
- transactions**
 - rolling back, 1211-1212, 1247
 - starting, 1201-1203, 1236-1237
- Transfer() method, 1369, 1372**
- transient keyword, 543**
- TristateFalse constant (format parameter), CD:1855**
- TristateTrue constant (format parameter), CD:1855**
- TristateUseDefault constant (format parameter), CD:1855**
- troubleshooting**
 - e-mail error messages, 1360-1361
 - Server Side JavaScript Application Manager, 163-164
- true keyword, 544**
- truth tables, 210-212, 219-221**
- try block, 97**
- try...catch block, 98-100**
- try...catch statement, 97-99**
- try...catch...finally statement, 544-546**
- type attribute, 32**
- type conversions, 39, 56-57**
- type property**
 - accessing, 902-903
 - alert boxes, 1041-1042, 1059-1060, 1080-1081
 - Button object, 597
 - button 's type, 609-610
 - Checkbox object, 615
 - code, 795-796, 930-931, 949, 962, CD:1489
 - CSSRule object, CD:1787
 - CSSUnknownRule object, CD:1804
 - displaying 713-714, 992
 - Event object, 698, CD:1733
 - File object, CD:1845, CD:1860
 - FileUpload object, 746
 - Folder object, CD:1891
 - Hidden object, 793
 - HTMLAnchorElement object, CD:1481
 - HTMLButtonElement object, CD:1513
 - HTMLInputElement object, CD:1586
 - HTMLLIElement object, CD:1607
 - HTMLLinkElement object, CD:1609
 - HTMLObjectElement object, CD:1624
 - HTMLParamElement object, CD:1636
 - HTMLPListElement object, CD:1649
 - HTMLScriptElement object, CD:1655
 - HTMLSelectElement object, CD:1659
 - HTMLStyleElement object, CD:1670
 - HTMLTextAreaElement object, CD:1715
 - HTMLUListElement object, CD:1726
 - MimeType object, 899
 - Password object, 922
 - Radio object, 938
 - Reset object, 952
 - rule type reading, CD:1789
 - Select object, 975
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1064
- TypeError (runtime error), 99**
- TypeError (runtime exception error), 344**
- typeName argument (ActiveXObject object), 242**
- typeof unary operator, 546-547**

U

- unbound() method (VBArray object), 551**
- UID parameter, 171**
- uid parameter, 1238, 1256-1258, 1262**
- uid return value (toString() method), 1215, 1252, 1269**
- UIEvent object, CD:1761**
- UIEvent.detail property, CD:1762**
- UIEvent.initUIEvent() method, CD:1762**
- UIEvent.view property, CD:1763**
- unary negation operator (-), 49, 203-204**
- unary operators, 546-547**
- UNC (Universal Naming Convention), CD:1953**
- undefined property, 365, 381, 548**
- undefined values, 36**
- underscore (_), 37**
- unencoded characters, 334-335, 369-370**
- unescape() method, 366, 380, 548-549**
- unicode parameter (CreateTextFile() method), CD:1870**
- uniform resource identifiers (URI), 330, 334, 670-671s**
- Universal Naming Convention (UNC), CD:1953**
- universal time, 324-325**
- UniversalBrowserAccess privilege, 22**
- UniversalBrowserRead privilege, 23**
- UniversalBrowserWrite privilege, 23**
- UniversalFileRead privilege, 23**
- UniversalPreferencesRead privilege, 23**
- UniversalPreferencesWrite privilege, 23**
- UniversalSendMail privilege, 23**
- UNIX, data tainting, 20**
- Unload event, 91, 680**
- UNLOAD**
 - event (Event object), 699
 - property (Event object), 699
- unlock() method, 1301, 1305, 1312-1314, 1373, 1377**
- unnamed objects, CD:2016-2017**
- unnamed parameters, CD:1951, CD:2017-2018**
- Unnamed property, CD:1944, CD:1951**
- unshift() method, 42, 250, 269-270**
- UNSPECIFIED EVENT TYPE ERR constant (EventException object), CD:1741**
- untaint() method, 1084**
- unwatch() method, 633**
 - Anchor object, 564
 - Array object, 250, 270
 - Area object, 573
 - Boolean object, 274, 278
 - Button object, 598
 - Checkbox object, 615
 - code, 450, 457, 490-491, 611, 691
 - Connection object, 1200
 - Cursor object, 1220
 - Date object, 288, 325
 - database object, 1234
 - DbPool object, 1257
 - Document object, 637
 - Event object, 698
 - File object, 1275
 - FileUpload object, 746
 - Form object, 761
 - Frame object, 779
 - Hidden object, 793
 - History object, 799
 - Image object, 808
 - Layer object, 822
 - Link object, 865
 - Location object, 886
 - Lock() object, 1301
 - Math object, 409
 - MimeType object, 899
 - navigator object, 904
 - Number object, 440
 - Object object, 453
 - Option object, 915
 - Password object, 923
 - Plugin object, 934
 - project object, 1312
 - Radio object, 939
 - RegExp object, 467
 - Reset object, 953
 - Resultset object, 1345
 - Select object, 975
 - SendMail object, 1355
 - server object, 1373
 - Stproc object, 1388
 - String object, 500
 - Style object, 995
 - Submit object, 1030
 - Text object, 1046
 - Textarea object, 1063
 - Window object, 1087
- updateRow() method (Cursor object), 1220, 1231-1232**
- upload boxes, 748**
- URIError (runtime error), 99, 344**
- URIs (uniform resource identifiers), 330, 334, 670-671**

url property (HTMLDocument object),
CD:1525

URL property, 637, 692, 1109,
CD:1535-1536

URL variable (ServerVariables collection),
1326

URLEncode() method, 1369, 1372

usage information (scripts), CD:1950

useMap property

code, CD:1583-1584, CD:1599,
CD:1635

HTMLImageElement object, CD:1577

HTMLInputElement object, CD:1586

HTMLObjectElement object, CD:1624

user interface events, CD:1761, CD:1763

user interfaces, 154-156

user sessions, 171

userAgent property, 904, 913

UserDomain property, CD:1964, CD:1971

UserName property, CD:1964, CD:1972

UTC() method, 288, 326-327

UTF-8 standard Web site, 366

\uXXXX escape sequence, 36

V

-v command-line parameter, 161

ValidatePersonallInfo function, 135

validation, 98-101, 135

**ValidFrom field (ClientCertificate collec-
tion), 1320**

**ValidUntil field (ClientCertificate collec-
tion), 1320**

vAlign property

code, CD:1683, CD:1687, CD:1709,
CD:1714

HTMLTableCellElement object,
CD:1674

HTMLTableColElement object,
CD:1684

HTMLTableRowElement object,
CD:1703

HTMLTableSectionElement object,
CD:1710

value argument (Boolean object), 273

value property

accessing, 990-991

Attr object, CD:1404

attributes, CD:1405-1406

Button object, 598

button 's text, 612-613

Checkbox object, 615

code, 797, 920-921, 932, 951, 964

FileUpload object, 746

Hidden object, 793

HTMLButtonElement object, CD:1513

HTMLInputElement object, CD:1586

HTMLLIElement object, CD:1607

HTMLOptionElement object, CD:1641

HTMLParamElement object, CD:1649

HTMLSelectElement object, CD:1659

Option object, 915

Password object, 922

Radio object, 938

Reset object, 952

Submit object, 1030, 1043-1044

text areas, 1082-1083

text boxes, 1061-1062

Text object, 1046

Textarea object, 1064

valueOf() method

Array object, 271

Boolean object, 274, 279

code, 363-364, 458

Date object, 327

Function object, 353

Number object, 440

Object object, 453

RegExp object, 467

values, 1252. See also return values

assigning, 54

attributes, CD:1405-1406

call by value (functions), 77-78

cookies, 138-139

counter, CD:1782

drive type, CD:1835

float, CD:1782-1786

Folder.Attributes property,

CD:1892-1893

functions, 79

modifiers, 707

numbers, 34-35

primitive Boolean versus Boolean
object, 274

rect, CD:1783

reference points, 1290

returning, 60, 79-80

string, CD:1784-1786

Submit object, 1043-1044

text areas, 1066-1067, 1082-1083

text boxes, 76, 1049, 1061-1062

undefined, 36

variables, 37-38

valueType property, CD:1652

valueHTMLParamElement object),

CD:1649

var keyword, 37-38, 549-550

var x = new Array(n); format, 40

variables, 37, CD:1955. See also environment variables

- @_alpha, 193-194
- @_jscript, 194
- @_jscript build, 195
- @_jscript version, 196
- @_jscript_build, 195
- @_jscript_version, 195
- @_mac, 196
- @_mc680x0, 197
- @_PowerPC, 197
- @_win16, 198
- @_win32, 199
- @_x86, 199-200
- _ (underscore), 37
- ALL HTTP, 1324
- ALL RAW, 1324
- APPL MD PATH, 1324
- APPL PHYSICAL PATH, 1324
- AUTH PASSWORD, 1324
- AUTH TYPE 1324, 1386
- AUTH USER, 1324
- CERT COOKIE, 1324
- CERT FLAGS, 1324
- CERT ISSUER, 1325
- CERT KEYSIZE, 1325
- CERT SECRETKEYSIZE, 1325
- CERT SERIALNUMBER, 1325
- CERT SERVER ISSUER, 1325
- CERT SERVER SUBJECT, 1325
- CERT SUBJECT, 1325
- characters, 37
- CONTENT LENGTH, 1325
- CONTENT TYPE, 1325
- declaring and assigning, 37-38
- declaring in functions, 38
- GATEWAY INTERFACE, 1325
- global, 38-39
- HTTP ACCEPT, 1325
- HTTP ACCEPT LANGUAGE, 1325
- HTTP COOKIE, 1325
- HTTP HeaderName [edit, HTML brackets], 1325
- HTTP REFERER, 1325
- HTTP USER AGENT, 1325
- HTTPS, 1325

HTTPS (ssjs_getCGIVariable() function), 1386

HTTPS KEYSIZE, 1325, 1386

HTTPS SERVER ISSUER, 1325

HTTPS SERVER SUBJECT, 1325

INSTANCE ID, 1325

INSTANCE META PATH, 1325

letters, 37

local, 38

LOCAL ADDR, 1325

LOGON USER, 1325

naming, 37

number1, 54

number2, 54

PATH INFO, 1325, 1386

PATH TRANSLATED, 1325, 1387

QUERY STRING, 1325, 1387

REMOTE ADDR, 1325, 1387

REMOTE HOST, 1326, 1387

REMOTE USER, 1326, 1387

REQUEST METHOD, 1326, 1387

scope, 38-39

SCRIPT NAME, 1326, 1387

server, 177-178

SERVER NAME, 1326, 1387

SERVER PORT, 1326, 1387

SERVER PORT SECURE, 1326

SERVER PROTOCOL, 1326, 1387

SERVER SOFTWARE, 1326

SERVER URL (ssjs_getCGIVariable() function), 1387

ServerVariables collection, 1324-1326

type conversions, 39

URL, 1326

values, assigning, 37-38

var keyword, 38

Windows environment, CD:1955

variables@_PowerPC, 197

VBAArray object, 551-552, CD:1829

VBAArray.dimensions() method, 552-553

VBAArray.getItem() method, 553-554

VBAArray.lbound() method, 555-556

VBAArray.toArray() method, 556-557

VBAArray.ubound() method, 557-558

vehicle objects, 8-9

Vendor error, 167

verification of origin, 19

verifying

authorization, 1331

dates, 17

locks, 1302

versatility (client-side and server-side), 24-25
version property, CD:1568-1569, CD:1942
Version property (WScript object), CD:1929
versions of JScript version, 196
verticalAlign property (Style object), 650, 663, 689
view property (UIEvent object), CD:1761
ViewCSS object, CD:1818
ViewCSS.getComputedStyle() method, CD:1818
visibility property, 822, 862
visible property, 1123-1125, 1145-1146, 1158, 1167-1170
Visual Basic, converting arrays to JScript arrays, CD:1829
vLink property, CD:1508, CD:1511-1512
vlinkColor property, 637, 692-693, 1109
void operator, 559
volatile keyword, 559-560
volume names, CD:1842
VolumeName property, CD:1833, CD:1842
vspace property
 code, 819, CD:1496, CD:1584, CD:1635
 HTMLAppletElement object, CD:1490
 HTMLImageElement object, CD:1577
 HTMLObjectElement object, CD:1624
 Image object, 808

W

w option (open() method), 1276, 1286
w+ option (open() method), 1276, 1286
W3C DOM Recommendation Web site, 149
warnings (cookies), 136
watch() method, 635-636, 915, 1087
 Anchor object, 564
 Area object, 573, 596-597
 Array object, 250, 271-272
 Boolean object, 274, 279-280
 Button object, 598
 Checkbox object, 615
 code, 451-452, 459, 492, 613-614, 693-694
 Connection object, 1200
 Cursor object, 1220
 Date object, 288, 328
 database object, 1234
 DbPool object, 1257
 Document object, 637

 Event object, 698
 File object, 1275
 FileUpload object, 746
 Form object, 761
 Frame object, 779
 Hidden object, 793
 History object, 799
 Image object, 808
 Layer object, 822
 Link object, 865
 Location object, 886
 Math object, 409
 navigator object, 904
 Number object, 440
 Object object, 453
 Password object, 923
 Plugin object, 934
 project object, 1312
 Radio object, 939
 RegExp object, 467
 Reset object, 953
 Resultset object, 1345
 Select object, 975
 SendMail object, 1355
 server object, 1373
 Stproc object, 1388
 String object, 500
 Style object, 995
 Submit object, 1030
 Text object, 1046
 Textarea object, 1063

watch() property (MimeType object), 899

Web-based applications, 27

Web browsers

 AOL, 108
 client-side syntax, 563
 content.type of files, 1274
 controls, 116-120
 DHTML, 151-154
 events, 90-91
 files, 1177
 functionality extended, 115
 HotJava, 108, 115
 Internet Explorer, 111-114
 Internet Explorer 5.5, 111, 114
 JavaScript, 110-111
 JScript, 112-113
 layers, 153-154
 LiveConnect, 120-123
 Netscape Navigator, 108-111
 Netscape Navigator 2.02, 109

- Netscape Navigator 6, 110
- Opera, 108-109, 114
- OS/2, 109
- supporting, 108-109
- Web File Path application setting (Server Side JavaScript Application Manager), 164**
- Web pages, 26, 126-129**
- Web sites**
 - ad tags, 18
 - ESPN, 139
 - iPlanet Enterprise, 157
 - JavaScript general information, 28-29
 - JavaScript reference resources, 29-30
 - Microsoft, 118, 184, CD:1824, CD:1927
 - Microsoft Developer Network, 28
 - Netscape DevEdge, 21-22
 - Netscape Navigator DevEdge, 111
 - Object Central, 10
 - Opera, 109, 114
 - plug-ins array, 118
 - UTF-8 standard, 366
 - W3C DOM Recommendation, 149
 - XML, 155
- web.html file, 141**
- which property, 698, 718**
- while loops, 67-68, 73, 283**
- while statement, 560-561**
- whiteSpace property, 650, 663, 689, 996, 1028**
- width parameter, 1135, 1190**
- width property**
 - accessing, 974
 - code, 820-821, 1029, CD:1497, CD:1565, CD:1567-1568
 - Event object, 698
 - HTMLAppletElement object, CD:1490
 - HTMLHRElement object, CD:1565
 - HTMLIFrameElement object, CD:1570
 - HTMLImageElement object, CD:1577
 - HTMLObjectElement object, CD:1624
 - HTMLPreElement object, CD:1652
 - HTMLTableCellElement object, CD:1674
 - HTMLTableColElement object, CD:1684
 - HTMLTableElement object, CD:1689
 - Image object, 808
 - Style object, 650, 663, 689, 996
- win16 processor, code to alert when on, 198**
- win32 system, code to alert when on, 199**
- WINDIR (Windows environment variable), CD:1955**
- Window object, 12, 19, 1085-1090, 1134-1136**
- window property, 778, 792-793, 1088, 1173**
- window.alert() method, 1090**
- window.atob() method, 1091**
- window.back() method, 1092**
- window.blur() method, 1092**
- window.btoa() method, 1094**
- window.captureEvents() method, 1094-1096**
- window.clearInterval() method, 1096-1097**
- window.clearTimeout() method, 1098**
- window.close() method, 1099**
- window.closed property, 1100-1101**
- window.confirm() method, 1102**
- window.crypto property, 1103**
- window.crypto.random() method, 1104**
- window.crypto.signText() method, 1104-1105**
- window.defaultStatus property, 1105-1106**
- window.disableExternalCapture() method, 1106**
- window.document property, 1107-1109**
- window.enableExternalCapture() method, 1110**
- window.find() method, 1111**
- window.focus() method, 1112**
- window.forward() method, 1113, 1171**
- window.frames property, 1114**
- window.frames.length property, 1115**
- window.handleEvent() method, 1115**
- window.history property, 1116-1117**
- window.home() method, 1118**
- window.innerHeight property, 1118**
- window.innerWidth property, 1120**
- window.length property, 1121**
- window.location property, 1122**
- window.locationbar property, 1122**
- window.locationbar.visible property, 1123**
- window.menubar property, 1123-1124**
- window.menubar.visible property, 1124**
- window.moveBy() method, 1125**
- window.moveTo() method, 1126**
- window.name property, 1127**
- window.offscreenBuffering property, 1129**
- window.onBlur event handler, 1129**
- window.onDragDrop event handler, 1130**
- window.onError event handler, 1130-1131**
- window.onFocus event handler, 1131**

- window.onLoad event handler, 1132**
- window.onMove event handler, 1132**
- window.onResize event handler, 1133**
- window.onUnload event handler, 1133**
- window.open() method, 1134**
- window.opener property, 1137**
- window.outerHeight property, 1139**
- window.outerWidth property, 1140**
- window.pageXOffset property, 1142**
- window.pageYOffset property, 1143**
- window.parent property, 1144**
- window.personalbar property, 1144-1145**
- window.personalbar.visible property, 1145**
- window.print() method, 1146**
- window.prompt() method, 1147**
- window.releaseEvents() method, 1148-1149**
- window.resizeBy() method, 1150**
- window.resizeTo() method, 1151**
- window.routeEvent() method, 1152-1155**
- window.screenX property, 1154-1155**
- window.screenY property, 1155**
- window.scroll() method, 1156**
- window.scrollbars property, 1157**
- window.scrollbars.visible property, 1158**
- window.scrollBy() method, 1158-1159**
- window.scrollTo() method, 1160**
- window.self property, 1161**
- window.setHotKeys() method, 1162**
- window.setInterval() method, 1162-1163**
- window.setResizable() method, 1164**
- window.setTimeout() method, 1164**
- window.setZOptions() method, 1165**
- window.status property, 1166**
- window.statusbar property, 1167**
- window.statusbar.visible property, 1167**
- window.stop() method, 1168**
- window.toolbar property, 1169**
- window.toolbar.visible property, 1169**
- window.top property, 1170**
- window.unwatch() method, 1171**
- window.watch() method, 1172**
- window.window property, 1173**
- windows**
 - child code, 144-145
 - client-side scripting, 139-145
 - close() method, 1099-1100
 - closed property, 1101-1102
 - Debugging, 166
 - document property methods, 1109-1110
 - focus, 1092-1093
 - focus() method, 1112-1113

- information, sending between, 142
- moveTo() method, 1127
- name property, 1128
- open() method, 1136-1137
- opener property, 1138-1139
- pageXOffset property, 1142-1143
- pageYOffset property, 1143-1144
- parent code, 142, 144
- pop-up (for games), 139
- resizeBy() method, 1150-1151
- resizeTo() method, 1152
- scroll() method, 1156-1157
- scrollBy() method, 1159-1160
- scrollTo() method, 1160-1161
- self property, 1161-1162
- setResizable() method, 1164
- spaces between features, 142
- Trace Information, 1273
- window property, 1173

Windows, 20, CD:1955, CD:2000, CD:2011-2013

Windows Script Host (WSH), 7, 16, 27-28, 179-189, CD:1936-1937

Windows scripting, 179

WindowStyle property, CD:2005, CD:2011 with statement, 75-76, 56-5621

WorkingDirectory property, CD:2005, CD:2012

write() function, 1398-1399

write() method, 167, 177

- code, 1295-1296, 1335, 1344, CD:1536, CD:1921-1922

- Document object, 637

- File object, 1275

- HTMLDocument object, CD:1526

- Response object, 1335

- text, 694-695

- TextStream object, CD:1912

- window.document property, 1108

WriteBlankLines() method, CD:1912, CD:1922

writeByte() method, 1275, 1296-1297

WriteLine() method, CD:1912, CD:1923

writeln() method

- code, CD:1537

- Document object, 637

- File object, 1275

- files, 1297-1298

- HTMLDocument object, CD:1526

- text, 695-696

- window.document property, 1108

writing

- binary data to files, 1296-1297
- carriage returns to files, 1297-1298
- cookie attributes, 139
- cookies, 139
- information on forms, 128-131
- strings to files, 1297-1298

WRONG DOCUMENT ERR constant (DOMException object), CD:1429

WScript object, 182-183, CD:1929

WScript parameters, CD:1946-1947

WScript.Arguments property, CD:1930

WScript.ConnectObject() method, CD:1931

WScript.CreateObject() method, CD:1932

WScript.DisconnectObject() method, CD:1933

WScript.Echo() method, CD:1933

wscript.exe file, 185

WScript.FullName property, CD:1934

WScript.GetObject() method, CD:1934

WScript.Interactive property, CD:1935

WScript.Name property, CD:1936

WScript.Path property, CD:1937

WScript.Quit() method, CD:1937

WScript.ScriptFullName property, CD:1938

WScript.ScriptName property, CD:1938

WScript.Sleep() method, CD:1939

WScript.StdErr property, CD:1939

WScript.StdIn property, CD:1940

WScript.StdOut property, CD:1941

WScript.Version property, CD:1942

WScript.WshArguments property, CD:1942

WScript.WshController object, CD:1943

WScript.WshNetwork object, CD:1943

WScript.WshShell object, CD:1944

.wsf file extension, 180

WSH (Windows Script Host), 7, 16, 27-28, 179-189, CD:1936-1937

WshArguments object, CD:1944

WshArguments property (WScript object), CD:1929

WshArguments.Count property, CD:1945

WshArguments.Count() method, CD:1946

WshArguments.Item() method, CD:1946

WshArguments.Length property, CD:1947

WshArguments.Named property, CD:1948

WshArguments.ShowUsage() method, CD:1949

WshArguments.Unnamed property, CD:1950

WshArguments.WshNamed object, CD:1951

WshArguments.WshUnnamed object, CD:1952

WshController object, CD:1952-1953, CD:1973

WshController.CreateScript() method, CD:1953

WshController.WshRemote object, CD:1954

WshEnvironment object, CD:1954-1956

WshEnvironment property (WshShell object), CD:1990

WshEnvironment.Count() method, CD:1956

WshEnvironment.Item() method, CD:1957

WshEnvironment.Length() property, CD:1957

WshEnvironment.Remove() method, CD:1958

WshNamed object, CD:1960

WshNamed.Count() method, CD:1961

WshNamed.Exists() method, CD:1961-1962

WshNamed.Item property, CD:1962

WshNamed.length property, CD:1963

WshNetwork object, 183-184, CD:1963-1964

WshNetwork.AddPrinterConnection() method, CD:1964

WshNetwork.AddWindowsPrinterConnection() method, CD:1965

WshNetwork.ComputerName property, CD:1966

WshNetwork.EnumNetworkDrives() method, CD:1967

WshNetwork.EnumPrinterConnection() method, CD:1967

WshNetwork.MapNetworkDrive() method, CD:1968

WshNetwork.RemoveNetworkDrive() method, CD:1969

WshNetwork.RemovePrinterConnection() method, CD:1970

WshNetwork.SetDefaultPrinter() method, CD:1970

WshNetwork.UserDomain property, CD:1971

WshNetwork.UserName property, CD:1971-1972

WshRemote object, CD:1972-1973
WshRemote.End event, CD:1973
WshRemote.Error event, CD:1974
WshRemote.Error property, CD:1974
WshRemote.Execute() method, CD:1975
WshRemote.Start event, CD:1976
WshRemote.Status property, CD:1977
WshRemote.Terminate() method, CD:1978
WshRemote.WshRemoteError object, CD:1976
WshRemoteError object, CD:1979
WshRemoteError.Character property, CD:1980
WshRemoteError.Description property, CD:1980-1981
WshRemoteError.Line property, CD:1981
WshRemoteError.Number property, CD:1982
WshRemoteError.Source property, CD:1983
WshRemoteError.SourceText property, CD:1984
WshScriptExec object, CD:1985
WshScriptExec.Status property, CD:1985
WshScriptExec.StdErr property, CD:1986
WshScriptExec.StdIn property, CD:1987
WshScriptExec.StdOut property, CD:1988
WshScriptExec.Terminate() method, CD:1989
WshShell object, 183, CD:1990-1991
WshShell.AppActivate() method, CD:1991
WshShell.CreateShortcut() method, CD:1991
WshShell.CurrentDirectory property, CD:1992
WshShell.Environment property, CD:1993
WshShell.Exec() method, CD:1993
WshShell.ExpandEnvironmentStrings() method, CD:1994
WshShell.LogEvent() method, CD:1995
WshShell.Popup() method, CD:1996
WshShell.RegDelete() method, CD:1997
WshShell.RegRead() method, CD:1998
WshShell.RegWrite() method, CD:1998
WshShell.Run() method, CD:1999-2000
WshShell.SendKeys() method, CD:2000-2001
WshShell.SpecialFolders property, CD:2002

WshShell.WshEnvironment object, CD:2003-2004
WshShell.WshScriptExec object, CD:2004
WshShell.WshShortcut object, CD:2003
WshShell.WshSpecialFolders object, CD:2004
WshShell.WshUrlShortcut object, CD:2003
WshShortcut object, CD:2004-2005
WshShortcut property (WshShell object), CD:1990
WshShortcut.Arguments property, CD:2005
WshShortcut.Description property, CD:2006
WshShortcut.FullName property, CD:2007
WshShortcut.Hotkey property, CD:2007-2008
WshShortcut.IconLocation property, CD:2008
WshShortcut.Save() method, CD:2009
WshShortcut.TargetPath property, CD:2010
WshShortcut.WindowStyle property, CD:2010
WshShortcut.WorkingDirectory property, CD:2011
WshSpecialFolders object, CD:2012-2013
WshSpecialFolders property (WshShell object), CD:1990
WshSpecialFolders.Count() method, CD:2013-2014
WshSpecialFolders.Item property, CD:2014
WshSpecialFolders.Length property, CD:2015
WshUnnamed object, CD:2015-2017
WshUnnamed.Count() method, CD:2017
WshUnnamed.Item property, CD:2018
WshUnnamed.Length property, CD:2019
WshUrlShortcut object, CD:2019-2020
WshURLShortcut property (WshShell object), CD:1990
WshUrlShortcut.FullName property, CD:2020
WshUrlShortcut.Save() method, CD:2021
WshUrlShortcut.TargetPath property, CD:2022

X-Y-Z

- x property (Anchor object), 564**
- XML, 155, CD:1416**
- XML-based User Interface Language (XUL), 154-154**
- XOR (>) bitwise operator, 60**
- XUL (XML-based User Interface Language), 154-156**
- \XX escape sequence, 36**
- \XXX escape sequence, 36**

- y property (Anchor object), 564, 571**
- year argument (Date object), 286**
- year argument (UTC() method), 326**

- z-lock parameter (Window object), 1136**
- zIndex property, 864**

Hey, you've got enough worries.

Don't let IT training be one of them.



Get on the fast track to IT training at InformIT,
your total Information Technology training network.



www.informit.com

SAMS

- Hundreds of timely articles on dozens of topics
- Discounts on IT books from all our publishing partners, including Sams Publishing
- Free, unabridged books from the InformIT Free Library
- “Expert Q&A”—our live, online chat with IT experts
- Faster, easier certification and training from our Web- or classroom-based training programs
- Current IT news
- Software downloads
- Career-enhancing resources

InformIT is a registered trademark of Pearson. Copyright ©2001 by Pearson.
Copyright ©2001 by Sams Publishing.

By opening this package, you are agreeing to be bound by the following agreement:

You may not copy or redistribute the entire CD-ROM as a whole. Copying and redistribution of individual software programs on the CD-ROM is governed by terms set by individual copyright holders.

The installer and code from the author(s) are copyrighted by the publisher and the author(s). Individual programs and other items on the CD-ROM are copyrighted or are under GNU license by their various authors or other copyright holders.

This software is sold as-is without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Neither the publisher nor its dealers or distributors assumes any liability for any alleged or actual damages arising from the use of this program. (Some states do not allow for the exclusion of implied warranties, so the exclusion may not apply to you.)

NOTE: This CD-ROM uses long and mixed-case filenames, requiring the use of a protected-mode CD-ROM Driver.

Installation Instructions

Windows

1. Insert the disc into your CD-ROM drive.
2. From the Windows desktop, double-click the My Computer icon.
3. Double-click the icon representing your CD-ROM drive.
4. Double-click on `start.exe`. Follow the on-screen prompts to finish the installation.

NOTE: If you have the AutoPlay feature enabled, `start.exe` will be launched automatically whenever you insert the disc into your CD-ROM drive.

Macintosh

1. Insert the disc into your CD-ROM drive.
2. Double-click the Pure JavaScript icon when it appears on your desktop.
3. Double-click on `start`. Follow the on-screen prompts to finish the installation.

Linux/Unix

These installation instructions assume that you have a passing familiarity with Unix commands and the basic setup of your machine. As Unix has many flavors, only generic commands are used. If you have any problems with the commands, please consult the appropriate manual page or your system administrator.

Insert the disc into your CD-ROM drive.

If you have a volume manager, mounting of the CD-ROM will be automatic. If you don't have a volume manager, you can mount the CD-ROM by typing

```
mount -tiso9660 /dev/cdrom /mnt/cdrom
```

NOTE: `/mnt/cdrom` is just a mount point, but it must exist when you issue the mount command. You may also use any empty directory for a mount point if you don't want to use `/mnt/cdrom`.

Open the `readme` file for descriptions and installation instructions.