

WebLOAD JavaScript Reference Manual

Version 6.0

RadView Software

The software supplied with this document is the property of RadView Software and is furnished under a licensing agreement. Neither the software nor this document may be copied or transferred by any means, electronic or mechanical, except as provided in the licensing agreement. The information in this document is subject to change without prior notice and does not represent a commitment by RadView Software or its representatives.

WebLOAD JavaScript Reference Manual

© Copyright 1998-2003 by RadView Software. All rights reserved.

November 1998, RadView Publication Number WL-0898-PG03

Revised: June, 2003

WebLOAD, TestTalk, Authoring Tools, ADL, AppletLoad, and WebExam, are trademarks or registered trademarks of RadView Software IBM, and OS/2 are trademarks of International Business Machines Corporation. Microsoft Windows, Microsoft Windows 95, Microsoft Windows NT, Microsoft Word for Windows, Microsoft Internet Explorer, Microsoft Excel for Windows, Microsoft Access for Windows and Microsoft Access Runtime are trademarks or registered trademarks of Microsoft Corporation. SPIDERSESSION is a trademark of NetDynamics. UNIX is a registered trademark of AT&T Bell Laboratories. Solaris, Java and Java-based marks are registered trademarks of Sun Microsystems, Inc. HP-UX is a registered trademark of Hewlett-Packard. SPARC is a registered trademark of SPARC International, Inc. Netscape Navigator and LiveConnect are registered trademarks of Netscape Communications Corporation. Any other trademark name appearing in this book is used for editorial purposes only and to the benefit of the trademark owner with no intention of infringing upon that trademark.



For product assistance or information, contact:

Toll free in the US:	1-888-RADVIEW
Fax:	(781) 238-8875
World Wide Web:	http://www.radview.com

North American Headquarters:

RadView Software Inc.
7 New England Executive Park
Burlington, MA 01803
Phone: (781) 238-1111
Email: sales@radview.com

International Headquarters:

RadView Software Ltd.
2 Habarzel Street
Tel Aviv Israel 69710
Phone: +972-3-765-0555
Fax: +972-3-647-1406

Table of Contents

Object Model Table List	xviii
How to use this book	xix
Before you begin.....	xix
About this guide	xix
WebLOAD documentation	xx
Where to Get More Information	xxi
On-line Help	xxi
Technical Support	xxi
Technical Support Web Site	xxi
Chapter 1: Introduction to JavaScript Agendas	3
1.1 What are Agendas?	4
1.2 WebLOAD Agendas work with an extended version of the standard DOM	5
1.2.1 What is the Document Object Model?	5
1.2.2 Understanding the DOM structure	6
1.2.3 DOM objects commonly used in an Agenda	9
1.2.4 WebLOAD extension set.....	10
1.3 When would I edit the JavaScript in my Agendas?	12
1.4 Accessing Agenda components	13
1.5 Editing the JavaScript code in an Agenda	16
1.5.1 Accessing the JavaScript code within the Agenda Tree	16
1.5.2 Using the IntelliSense JavaScript Editor	17
Chapter 2: WebLOAD Actions, Objects, and Functions	19
action (property).....	19

Actions	20
Add() (method)	21
addProperty() (method)	22
AdjacentText (property)	23
Alt (property)	23
area (object)	24
Automatic State Management for HTTP Protocol Mode ...	25
AutoNavigate() (action)	26
Back() (action)	27
BeginTransaction() (function)	28
bitrate (property)	30
Browser configuration components	30
BrowserCache (property)	33
Button (object)	34
cell (object)	35
cellIndex (property)	37
Checkbox (object)	38
checked (property)	39
ClearAll() (method)	40
ClearDNSCache() (method)	41
ClearSSLCache() (method)	42
ClientNum (variable)	43
Close() (function)	45
CloseConnection() (method)	46
Collections	47
cols (property)	48
Compare() (method)	49
CompareColumns (property)	51
CompareRows (property)	52
connectionBandwidth (property)	53
ConnectionSpeed (property)	54
ContainerTag (property)	55
ContainingMap (property)	55
content (property)	56
Coords (property)	57
CopyFile() (function)	57
CreateDOM() (function)	59
CreateTable() (function)	60
currentPosition (property)	62

currentStreamName (property)	62
Data (property).....	63
DataFile (property).....	65
DefaultAuthentication (property)	66
defaultchecked (property)	68
defaultselected (property)	68
defaultvalue (property).....	69
delete() (method)	70
Details (property)	72
Dialog box properties	73
DisableSleep (property)	75
Div (object).....	77
DNSUseCache (property)	78
document (object)	79
duration (property)	80
Dynamic Object Recognition (DOR) components.....	81
element (object)	83
encoding (property).....	85
EndTransaction() (function)	85
Erase (property).....	87
ErrorMessage() (function)	89
EvaluateScript() (function)	90
event (property)	91
ExceptionEnabled (property)	92
ExpectedDocument (property)	93
ExpectedDOMID (property)	94
ExpectedID (property).....	94
ExpectedLocation (property).....	95
ExpectedName (property).....	96
ExpectedText (property)	97
ExpectNavigation() (method)	97
File (object).....	99
File management functions	100
fileName (property)	100
FindObject() (method).....	102
form (object).....	102
FormData (property)	104
Forward() (action)	106
frames (object).....	107

GeneratorName() (function)	108
Get() (method).....	110
Get() (addition method)	110
Get() (transaction method)	111
GetCurrentValue() (method).....	114
GetFieldValue() (method).....	115
GetFieldValueInForm() (method)	116
GetFormAction() (method)	117
GetFrameByUrl() (method).....	118
GetFrameUrl() (method).....	119
GetHeaderValue() (method).....	120
GetHost() (method)	121
GetHostName() (method).....	122
GetIPAddress() (method)	123
GetLine() (function)	124
GetLinkByName() (method)	127
GetLinkByUrl() (method)	128
GetMessage() (method).....	129
GetOperatingSystem() (function)	130
GetPortNum() (method)	131
GetQSFieldValue() (method).....	132
GetSeverity() (method).....	133
GetStatusLine() (method).....	134
GetStatusNumber() (method).....	135
GetUri() (method).....	136
hash (property).....	137
Head() (method).....	137
Header (property).....	138
HistoryLimit (property).....	140
host (property).....	141
hostname (property).....	141
href (property)	142
httpEquiv (property).....	143
id (property).....	143
Identification variables and functions	146
IdentifyObject() (method)	147
Image (object)	148
IncludeFile() (function)	150
Index (property).....	151
InfoMessage() (function)	152

InnerHTML (property)	153
InnerImage (property)	154
InnerLink (property).....	155
InnerText (property)	155
InputButton (object).....	157
InputCheckbox (object)	158
InputFile (object)	159
InputImage (object).....	160
InputPassword (object)	161
InputRadio (object).....	162
InputText (object)	164
KeepAlive (property)	165
key (property).....	166
language (property).....	167
length (property)	167
link (object)	169
load() (method)	170
load() and loadXML() method comparison	172
LoadGeneratorThreads (property)	173
loadXML() (method)	175
location (object)	176
map (object).....	178
MatchBy (property)	178
MaxLength (property).....	180
Message functions.....	180
method (property)	182
MultilIPSupport (property).....	183
Name (property).....	184
Navigate() (action)	186
NTUserName, NTPassWord (properties)	187
Num() (method)	188
ObjectProperty[] (property)	189
Objects	189
OnClick (property).....	190
OnMouseOver (property)	191
Open() (function).....	193
OpenStream() (method).....	194
option (object)	195
OuterLink (property).....	196

Outfile (property)	197
PassWord (property)	198
pathname (property).....	199
Pause() (method)	200
Play() (method)	200
port (property)	202
Post() (method)	202
Prepare() (method).....	205
ProbingClientThreads (property)	206
protocol (property).....	208
Proxy, ProxyUserName, ProxyPassWord (properties).....	209
Radiobutton (object).....	210
Range() (method).....	211
ReceiveTimeout (property).....	212
RedirectionLimit (property)	212
Refresh() (action)	213
ReportEvent() (function).....	214
ReportLog() (method).....	215
ReportUnexpectedRows (property).....	216
RequestRetries (property).....	217
Reset	218
Reset (object)	218
Reset() (method)	219
Resume() (method)	220
RoundNum (variable)	221
row (object)	223
rowIndex (property)	225
SaveSource (property)	226
SaveTransaction (property).....	227
SaveWSTransaction (property).....	228
script (object).....	229
search (property).....	230
Seed() (method)	231
Select	231
Select (object).....	231
Select() (method).....	232
selected (property)	233
selectedIndex (property).....	234
SendCounter() (function).....	234
SendMeasurement() (function).....	235
SendTimer() (function)	236

Set() (method).....	237
Set() (addition method)	237
Set() (cookie method).....	238
SetFailureReason() (function).....	239
SetTimer() (function).....	241
setType() (method)	242
setValue() (method)	243
SetWindow() (action)	244
SevereErrorMessage() (function).....	245
Shape (property).....	246
Size (property)	247
Sleep() (function)	247
SleepDeviation (property)	250
SleepRandomMax (property).....	251
SleepRandomMin (property).....	252
Span (object)	254
src (property)	255
SSL Cipher Command Suite	256
SSLBitLimit (property).....	257
SSLCipherSuiteCommand() (function).....	258
SSLClientCertificateFile, SSLClientCertificatePassword (properties)	260
SSLCryptoStrength (property)	262
SSLDisableCipherID() (function).....	263
SSLDisableCipherName() (function).....	265
SSLEnableCipherID() (function).....	266
SSLEnableCipherName() (function).....	267
SSLGetCipherCount() (function).....	268
SSLGetCipherID() (function).....	269
SSLGetCipherInfo() (function)	270
SSLGetCipherName() (function).....	271
SSLGetCipherStrength() (function).....	272
SSLUseCache (property).....	273
SSLVersion (property)	275
state (property)	277
Stop() (method).....	278
string (property)	278
Submit (object).....	279
SyncDOM() (method).....	280
SynchronizationPoint() (function).....	281

table (object)	284
TableCell (object)	285
TableCompare (object)	286
TableCompare() (constructor)	287
tagName (property)	289
target (property)	290
text (property)	291
TextArea (object)	292
TimeoutSeverity (property)	293
Timing functions	295
title (property)	296
Transaction verification components	297
TransactionTime (property)	298
type (property)	300
UIContainer (object)	301
Url (property)	302
UseHistory (property)	304
User-defined global properties	304
UserAgent (property)	306
UserName (property)	307
UsingTimer (property)	308
Validate() (method)	309
value (property)	310
VCUniqueID() (function)	312
Verification (property)	313
Verification Test Components	315
Verification Test Property List: Global and Page Level	318
VerificationFunction() (user-defined) (function)	320
Version (property)	322
WarningMessage() (function)	323
window (object)	324
wlBrowser (object)	325
wlClick() (action)	326
wlClear() (method)	328
wlClose() (action)	329
wlCookie (object)	330
wlException (object)	331
wlException() (constructor)	333
wlGeneratorGlobal (object)	334

wlGet() (method).....	335
wlGetAllForms() (method).....	336
wlGetAllFrames() (method).....	337
wlGetAllLinks() (method).....	338
wlGlobals (object)	339
wlHeader (object).....	340
wlHtml (object)	342
wlHttp (object).....	342
wlLocals (object)	343
wlMediaPlayer (object).....	344
wlMediaPlayer() (constructor)	345
wlMeta (object)	346
wlMouseDown() (action)	347
wlMouseOver() (action).....	348
wlMouseUp() (action).....	349
wlMultiSelect() (action).....	350
wlOutputFile (object).....	351
wlOutputFile() (constructor)	354
wlRand (object).....	355
wlSearchPair (object).....	356
wlSelect() (action)	358
wlSet() (method)	359
wlSource (property).....	360
wlStatusLine (property).....	361
wlStatusNumber (property)	362
wlSubmit() (action).....	362
wlSystemGlobal (object)	363
wlTable (object).....	364
wlTarget (property).....	366
wlTypeIn() (action)	366
wlVersion (property).....	368
WLXmlDocument() (constructor)	368
wlXmIs (object)	370
Write() (method).....	372
WriteIn() (method).....	373
WSComplexObject (object).....	374
WSComplexObject() (constructor)	375
WSGetSimpleValue() (function).....	376
WSWebService (object).....	377
WSWebService() (constructor)	379

XMLDocument (property).....	380
Appendix A: WebLOAD-supported SSL Protocol Versions.....	381
SSL handshake combinations	381
SSL protocols—complete list	382
128-bit encryption.....	382
56-bit encryption.....	385
40-bit encryption.....	387
0-bit encryption.....	388
Appendix B: WebLOAD-supported XML DOM Interfaces	391
Table B-1: XML Document Interface Properties.....	392
Table B-2: XML Document Interface Methods	393
Table B-3: Node Interface Properties	394
Table B-4: Node Interface Methods	395
Table B-5: Node List Interface.....	396
Table B-6: NamedNodeMap Interface.....	397
Table B-7: ParseError Interface	398
Table B-8: Implementation Interface	398
Appendix C: WebLOAD Internet Protocols Reference	399
wiFTP Object	400
wiFTP Properties.....	400
Data	400
DataFile	400
document.....	400
Outfile	400
PassiveMode	401
PassWord	401
Size	401
StartByte	401
TransferMode	402
UserName	402
wiFTP Methods	402
Append().....	402
AppendFile()	403
ChangeDir()	403
ChFileMod().....	403
ChMod().....	404
Delete().....	404
DeleteFile()	404
Dir().....	405
Download()	405
DownloadFile()	405

GetCurrentPath()	406
GetStatusLine()	406
ListLocalFiles()	406
Logoff()	407
Logon()	407
MakeDir()	407
RemoveDir()	407
Rename()	408
SendCommand()	408
Upload()	408
UploadFile()	409
UploadUnique()	409
WLFtp()	409
FTP Sample Code	410
wIIMAP Object	413
wIIMAP Properties	413
CurrentMessage	413
CurrentMessageID	413
document	413
Mailbox	414
MaxLines	414
Outfile	414
PassWord	415
Size	415
UserName	415
wIIMAP Methods	415
Connect()	415
CreateMailbox()	416
Delete()	416
DeleteMailbox()	416
Disconnect()	416
GetMessageCount()	417
GetStatusLine()	417
ListMailboxes()	417
RecentMessageCount()	417
RenameMailbox()	418
Retrieve()	418
Search()	418
SendCommand()	420
SubscribeMailbox()	420
UnsubscribeMailbox()	420
WLImap()	420
IMAP Sample Code	421
wINNTP Object	425
wINNTP Properties	425
ArticleText	425

Attachments.....	425
AttachmentsEncoding.....	425
AttachmentsTypes.....	426
document.....	426
From.....	427
Group.....	427
MaxHeadersLength.....	427
Organization.....	427
Outfile.....	427
PassWord.....	427
References.....	428
ReplyTo.....	428
Size.....	428
Subject.....	428
To.....	428
UserName.....	429
wINNTP Methods.....	429
AddAttachment().....	429
Connect().....	429
DeleteAttachment().....	430
Disconnect().....	430
GetArticle().....	430
GetArticleCount().....	431
GetStatusLine().....	431
GroupOverview().....	431
ListGroup().....	431
PostArticle().....	432
SendCommand().....	432
WLNntp().....	433
NNTP Sample Code.....	433
wIPOP Object.....	436
wIPOP Properties.....	436
AutoDelete.....	436
document.....	436
Headers[].....	437
MaxLines.....	437
Outfile.....	437
PassWord.....	437
Size.....	438
UserName.....	438
wISource.....	438
wIPOP Methods.....	438
Connect().....	438
Delete().....	439
Disconnect().....	439
GetCurrentMessageID().....	439
GetMailboxSize().....	439

GetMessageCount().....	440
GetStatusLine()	440
Reset().....	440
Retrieve().....	440
SendCommand()	441
WLPop()	441
POP Sample Code.....	441
wISMTTP Object.....	444
wISMTTP Properties	444
Attachments	444
AttachmentsEncoding	444
AttachmentsTypes.....	445
Bcc.....	445
Cc	445
From	445
Message.....	446
ReplyTo	446
Size.....	446
Subject.....	446
To	446
Type.....	446
wISMTTP Methods.....	447
AddAttachment()	447
Connect().....	447
DeleteAttachment()	448
Disconnect()	448
Send().....	448
SendCommand()	448
Verify().....	449
WLSmtp()	449
SMTP Sample Code	449
wITCP Object.....	451
wITCP Properties	451
document.....	451
InBufferSize.....	451
LocalPort	451
NextPrompt	452
NextSize	452
OutBufferSize	452
Outfile	452
ReceiveMessageText.....	452
Size.....	453
Timeout.....	453
wITCP Methods.....	453
Connect().....	453
Disconnect()	454

Erase().....	454
Receive()	454
Send().....	455
WLTcp().....	455
TCP Sample Code	456
wITelnet Object.....	457
wITelnet Properties	457
document.....	457
NextPrompt.....	457
Outfile	457
ReceiveMessageText.....	458
Size.....	458
Timeout.....	458
wITelnet Methods	459
Connect().....	459
Disconnect()	459
Erase().....	459
Receive()	459
Send().....	460
WLTelnet().....	460
Telnet Sample Code	460
wIUDP Object.....	463
wIUDP Properties	463
document.....	463
InBufferSize	463
LocalHost.....	463
LocalPort.....	464
MaxDatagramSize	464
NumOfResponses	464
OutBufferSize	464
Outfile	464
ReceiveMessageText.....	465
RequestedPackets	465
Size.....	465
Timeout.....	465
wIUDP Methods	466
Bind()	466
Broadcast().....	466
Erase().....	466
Receive()	467
Send().....	467
UnBind().....	467
WLUdp().....	468
UDP Sample Code.....	468
Appendix D: HTTP Protocol Status Messages.....	471

D.1	Informational 1XX	472
D.2	Success 2XX	473
D.3	Redirection 3XX	476
D.4	Client Error 4XX	480
D.5	Server Error 5XX	484
Index	485

Object Model Table List

Table 1-1: DOM objects commonly used in Agendas	9
Table 1-2: WebLOAD DOM extension set highlights.....	10
Table 2-1: Dialog Box Property List.....	74
Table 2-2: load() and loadXML() comparison.....	172
Table A-1: SSL handshake combinations	381
Table B-1: XML Document Interface Properties.....	392
Table B-2: XML Document Interface Methods.....	393
Table B-3: Node Interface Properties.....	394
Table B-4: Node Interface Methods.....	395
Table B-5: Node List Interface	396
Table B-6: NamedNodeMap Interface.....	397
Table B-7: ParseError Interface	398
Table B-8: Implementation Interface	398
Table D-1: Informational <i>1XX</i> message set.....	472
Table D-2: Successful <i>2XX</i> message set	473
Table D-3: Redirectional <i>3XX</i> message set.....	476
Table D-4: Client Error <i>4XX</i> message set	480
Table D-5: Severe Error <i>5XX</i> message set	484

How to use this book

This book provides complete reference information for all objects, variables, and functions defined by WebLOAD for JavaScript Agendas.

Before you begin

Before you start to program Agendas, you should be familiar with WebLOAD and read *The WebLOAD User's Guide*.

About this guide

This book provides a detailed reference to the Document Object Model. The chapters are arranged as follows:

Chapter 1, *Introduction to JavaScript Agendas*

This chapter provides an introduction to the Document Object Model (DOM) and how to use it.

Chapter 2, *WebLOAD Actions, Objects, and Functions*

This chapter provides a comprehensive listing of all the DOM objects, properties, methods, functions, and variables.

Appendix A, *WebLOAD-supported SSL Protocol Versions*

This appendix provides a list of all supported SSL protocols and a table of the various handshake combinations possible.

Appendix B, *WebLOAD-supported XML DOM Interfaces*

This appendix provides a description of the XML DOM Document Interfaces supported by WebLOAD.

Appendix C, *WebLOAD Internet Protocols Reference*

This appendix provides detailed reference information on WebLOAD support for the following Internet protocols: FTP, IMAP, NNTP, POP, SMTP, TCP, Telnet and UDP.

Appendix D, *HTTP Protocol Status Messages*

This appendix provides a list of the HTTP Protocol Status messages that you may see over the course of a test session.

WebLOAD documentation

WebLOAD is supplied with the following documentation:

The WebLOAD™ User's Guide

Instructions for installing and using WebLOAD and for running tests.

Recording WebLOAD™ Agendas

Instructions for creating JavaScript Agendas for use in WebLOAD by recording your activities in a Web browser. No programming is necessary.

WebLOAD™ Programming Guide

Complete information on programming and editing JavaScript Agendas for use in WebLOAD.

WebLOAD JavaScript Reference Manual (this book)

Complete reference information on all JavaScript objects, variables, and functions used in test Agendas.

WebLOAD™ Resource Manager User's Guide

Instructions for installing and running the WebLOAD Resource Manager.

Core JavaScript Language Guide

Reference manual for the JavaScript language (published by Netscape Communications Corporation and distributed with permission). The section entitled *The Core JavaScript Language* describes the features used in WebLOAD programming.

The manuals are distributed with the WebLOAD software in online help format. The manuals are also supplied as Adobe Acrobat files that you can view and print using the Adobe Acrobat Reader. Install the Reader from the WebLOAD CD-ROM or download it from the Adobe Web site (<http://www.adobe.com>).

Where to Get More Information

This section contains information on how to obtain technical support from RadView worldwide, should you encounter any problems.

On-line Help

WebLOAD provides a comprehensive on-line help system with step-by-step instructions for common tasks.

You can press the F1 key on any open dialog box for an explanation of the options or select **Help | Contents** to open the on-line help contents and index.

Technical Support

For technical support in your use of WebLOAD, contact:

◆ North American Headquarters

e-mail: support@radview.com

Phone: 1-888-RADVIEW (1-888-728-8439) (Toll Free)

781-238-1111

Fax: 781-238-8875

◆ International Headquarters

e-mail: support@radview.com

Phone: +972-3-765-0555

Fax: +972-3-647-1406

Note: We encourage you to use e-mail for faster and better service.

When contacting technical support please include in your message the full name of the product (WebLOAD), as well as the version and build number.

Technical Support Web Site

The technical support pages on our Web site contains:

- ◆ FAQ (Frequently Asked / Answered Questions).
- ◆ RadView's Product Resource Center, where you can find prewritten test scripts, product information, industry related news, and the TestTalk newsletter, with hints and tips for customers and evaluators.

<http://www.radview.com/support/index.asp>

Introduction to JavaScript Agendas

The *WebLOAD JavaScript Reference Manual* provides a detailed description of the syntax and usage of the full set of WebLOAD JavaScript features, including the actions, objects, and functions used to create sophisticated test session Agendas. Note that most WebLOAD users do not need this level of detail to create effective testing sessions for their Web site. Agendas are usually recorded and edited through the Visual Agenda Authoring Tool (Visual AAT), a simple, intuitive interface that provides users with a comprehensive set of testing tools literally at their fingertips, though point-and-click or drag-and-drop convenience. The details in this manual are provided for the convenience of the more sophisticated programmers, who may wish to add specific, perhaps complex tailoring to their recorded Agendas.

This chapter provides a general introduction to JavaScript Agendas. It includes the following sections:

- ◆ What are Agendas?
- ◆ WebLOAD Agendas work with an extended version of the standard DOM
 - ◆ What is the Document Object Model?
 - ◆ Understanding the DOM structure
 - ◆ DOM objects commonly used in an Agenda
 - ◆ WebLOAD extension set
- ◆ When would I edit the JavaScript in my Agendas?
- ◆ Accessing Agenda components
- ◆ Editing the JavaScript code in an Agenda
 - ◆ Accessing the JavaScript code within the Agenda Tree
 - ◆ Using the IntelliSense JavaScript Editor

1.1 What are Agendas?

WebLOAD runs test sessions that simulate the actions of a real user through the use of Agenda files. Agendas are client programs that access the server you want to test. Users create Agendas by recording a series of typical activities with the application being tested through the Visual AAT. The Visual AAT automatically converts the user activities into Agenda programs. You do not need to know anything about writing Agendas to test an application with WebLOAD. No programming or editing skills are required to create or run a successful test session.

Agendas are created through the WebLOAD Visual Agenda Authoring Tool (Visual AAT). The Visual AAT operates in conjunction with a Web browser such as Microsoft's Internet Explorer. As a user navigates the test application in the browser, (for example, navigating between pages, typing text into a form, or clicking on the mouse), the Visual AAT records all user actions in an Agenda. During later Web site testing sessions, WebLOAD simulates every action of the original user and automatically handles all Web interactions, including parsing dynamic HTML, dynamic object recognition, and full support for all security requirements, such as user authentication or SSL protocol use.

A simple recorded Agenda is ideal if your WebLOAD test involves a typical sequence of Web activities. These activities are all recorded in your Agenda, and are represented in the Visual AAT GUI by an Agenda Tree, a set of clear, intuitive icons and visual devices arranged into a logical hierarchical sequence. Each of these activity icons actually represents a block of code within the underlying test Agenda. Agendas are constructed automatically out of 'building blocks' of test code, and most users create and run test sessions quite easily, without ever looking into those building blocks to see the actual code inside.

Some users prefer to manually edit the code of a recorded Agenda to create more complex, sophisticated test sessions. For example, for an Agenda to work with Java or COM components, a certain degree of programming is required. This manual documents the syntax of the JavaScript objects and functions available to programmers who wish to add more complex functionality to their Agendas.

Agendas are written in JavaScript. JavaScript is an object-oriented scripting language developed by Netscape Communications Corporation. JavaScript is best known for its use in conjunction with HTML to automate World Wide Web pages. However, JavaScript is actually a full-featured programming language that can be used for many purposes besides Web automation. WebLOAD has chosen JavaScript as the scripting language for test session Agendas. WebLOAD JavaScript Agendas combine the ease and simplicity of WebLOAD's visual, intuitive programming environment with the flexibility and power of JavaScript object-oriented programming.

For detailed information on using WebLOAD, including creating Agendas, running test sessions, and analyzing the results, see the *WebLOAD User's Guide*.

1.2 WebLOAD Agendas work with an extended version of the standard DOM

The Visual AAT operates in conjunction with a Web browser such as Microsoft's Internet Explorer. As you execute a sequence of HTTP actions in the browser, the Visual AAT records your actions in a JavaScript Agenda. All Web browsers rely on an extended Document Object Model, or DOM, for optimum handling of HTML pages. The standard browser DOM defines both the logical structure of HTML documents and the way a document is accessed and manipulated. WebLOAD Agendas use a standard browser DOM to access and navigate Internet Web pages, including Dynamic HTML and nested links and pages. To facilitate Web site testing, WebLOAD extends the standard browser DOM with many features, objects, and functions that expedite site testing and evaluation.

This section provides a brief overview of the standard DOM structure. Most of the information in this overview was provided by the World Wide Web Consortium (W3C), which develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding. For more information about the standard DOM structure and components, go to the following Web sites:

- ◆ <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/introduction.html>
- ◆ <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dom/domoverview.asp>
- ◆ <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/dhtmlrefs.asp>

1.2.1 What is the Document Object Model?

The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. The DOM defines the logical structure of documents and the way a document is accessed and manipulated. With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions—in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language.

Essentially, the DOM is a programming API for documents based on an object structure that closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

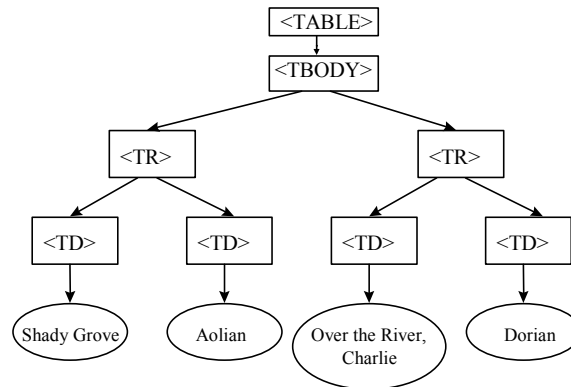
```

<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>

```

The following figure illustrates a typical DOM representation of this table:

DOM representation of a sample table



1.2.2 Understanding the DOM structure

In the DOM, documents have a logical structure that is very much like a tree; to be more precise, that is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one `doctype` nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document. However, the DOM does not specify that documents must be *implemented* as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term *structure model* to describe the tree-like representation of a document. We also use the term "tree" when referring to the arrangement of those information items which can be reached by using "tree-walking" methods; (this does not include attributes). One important property of DOM structure models is *structural isomorphism*: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, in accordance with the XML Information Set [Infoset].

Note: There may be some variations depending on the parser being used to build the DOM. For instance, the DOM may not contain white spaces in element content if the parser discards them.

The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense. Documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- ◆ the interfaces and objects used to represent and manipulate a document
- ◆ the semantics of these interfaces and objects - including both behavior and attributes
- ◆ the relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract data model, not by an object model. In an abstract data model, the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects that hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

The information in this section has been excerpted from the World Wide Web Consortium introduction to the DOM. For the complete text of the DOM overview, see <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/introduction.html>. The complete document is found at <http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/>.

Copyright © 1994-2001 [World Wide Web Consortium](#), ([Massachusetts Institute of Technology](#), [Institut National de Recherche en Informatique et en Automatique](#), [Keio University](#)).

All Rights Reserved. <http://www.w3.org/Consortium/Legal/>

W3C® DOCUMENT NOTICE AND LICENSE

Copyright © 1994-2001 [World Wide Web Consortium](#), ([Massachusetts Institute of Technology](#), [Institut National de Recherche en Informatique et en Automatique](#), [Keio University](#)).
All Rights Reserved. <http://www.w3.org/Consortium/Legal/>

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

- 1) A link or URL to the original W3C document.
- 2) The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [Date-of-document] [World Wide Web Consortium](#), ([Massachusetts Institute of Technology](#), [Institut National de Recherche en Informatique et en Automatique](#), [Keio University](#)). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>" (Hypertext is preferred, but a textual representation is permitted.)
- 3) *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the [Copyright FAQ](#)) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on April 05 1999 so as to account for the treatment of DTDs, schemas and bindings. See the [older formulation](#) for the policy prior to this date. Please see our [Copyright FAQ](#) for common questions about using materials from our site, including specific terms and conditions for packages like libwww, Amaya, and Jigsaw. Other questions about this notice can be directed to site-policy@w3.org.

[webmaster](#)
(last updated by reagle on 1999/04/99.)

1.2.3 DOM objects commonly used in an Agenda

On Internet Web sites, a simple HTML document may be constructed of a single page, or the document may be constructed of many nested pages, each one including multiple ‘child’ windows, in a recursive structure. Browser DOMs were designed to reflect this flexible approach.

When using the DOM, a single Web page document has a logical structure that resembles a single tree. In nested Web pages, each child window is simply one tree in a recursive forest of trees. The typical DOM is ideal for representing Internet Web page access because it provides a flexible, generic model that encompasses both the attributes of the object itself and its interfaces and behaviors. Typical DOM objects include:

- ◆ The document itself.
- ◆ The frames nested in an HTML page, together with any additional nested windows.
- ◆ The location information.
- ◆ The links, forms, and images on the page.
- ◆ The tables, scripts, XML Data Islands, and Meta objects on the page.
- ◆ Individual elements of a specific form or frame.

The following table provides a brief overview of the main DOM object components of a typical Web page.

Table 1-1: DOM objects commonly used in Agendas

Object	Description
window	The <code>window</code> object represents an open browser window. Typically, the browser creates a single <code>window</code> object when it opens an HTML document. However, if a document defines one or more frames the browser creates one <code>window</code> object for the original document and one additional <code>window</code> object (a <i>child window</i>) for each frame. The child window may be affected by actions that occur in the parent. For example, closing the parent window causes all child windows to close.
document	The <code>document</code> object represents the HTML document in a browser window, storing the HTML data in a parsed format. Use the <code>document</code> object to retrieve links, forms, nested frames, images, scripts, and other information about the document. By default, <code>document</code> used alone represents the document in the current window. You usually refer directly to the <code>document</code> ; the <code>window</code> part is optional and is understood implicitly.
frame	Each <code>frame</code> object represents one of the frames imbedded within a Web page. Frames and windows are essentially comparable. The recursive aspect of the DOM is implemented at this level. A window may contain a collection of frames. Each frame may contain multiple child windows, each of which may contain more frames that contain more windows, and so on.
location	The <code>location</code> object contains information on the current window URL.

Object	Description
link	A <code>link</code> object contains information on an external document to which the current document is linked.
form, element, and input	<p>A <code>form</code> object contains the set of elements and input controls (text, radio buttons, checkboxes, etc.) that are all components of a single form. Each <code>element</code> object stores the parsed data for a single HTML form element such as <code><INPUT></code>, <code><BUTTON></code>, or <code><SELECT></code>. Each <code>input</code> object stores the information defining one of the input controls in the form. Controls are organized by type, for example <code>input type=checkbox</code>.</p> <p>Forms enable client-side users to submit data to a server in a standardized format. A form is designed to collect the required data using a variety of controls, such as <code>INPUT</code> or <code>SELECT</code>. Users viewing the form fill in the data and then click the <code>SUBMIT</code> button to send it to the server. A script on the server then processes the data. Notice that the object syntax corresponds to a path through the DOM hierarchy tree, beginning at the root window and continuing until the specified item's properties.</p>
image	Each <code>image</code> object contains one of the embedded images found in a document.
script	A <code>script</code> object defines a script for the current document that will be interpreted by a script engine.
title	The <code>title</code> object contains the document title, stored as a text string.

1.2.4 WebLOAD extension set

WebLOAD has added the following extensions to the standard DOM properties and methods. This reference manual provides syntax specifications for these objects in *WebLOAD Actions, Objects, and Functions*, beginning on page 19.

Table 1-2: WebLOAD DOM extension set highlights

WebLOAD object extensions	Description
<code>wlCookie</code>	Sets and deletes cookies.
<code>wlException</code>	WebLOAD error management object.
<code>wlGeneratorGlobal</code> and <code>wlSystemGlobal</code> objects	Handles global values shared between Agenda threads or Load Generators.
<code>wlGlobals</code>	Manages global system and configuration values.
<code>wlHeader</code>	Contains the key/value pairs in the HTTP command headers that brought the document. (Get, Post, etc.)

WebLOAD object extensions	Description
wlHttp	Performs HTTP transactions and stores configuration property values for individual transactions.
wlLocals	Stores local configuration property values.
wlMediaPlayer	Supports streaming media applications.
wlMeta	Stores the parsed data for an HTML meta object.
wlOutputFile	Writes Agenda output messages to a global output file.
wlRand	Generates random numbers.
wlSearchPair	Contains the key/value pairs in a document's URL search strings.
wlTable, row, and cell objects	Contains the parsed data from an HTML table.
XML DOM objects	XML DOM object set that both accesses XML site information <i>and</i> generates new XML data to send back to the server for processing.

Web site testing usually means testing how typical user activities are handled by the application being tested. Are the user actions managed quickly, correctly, appropriately? Is the application responsive to the user's requests? Will the typical user be happy working with this application? When verifying that an application handles user activities correctly, WebLOAD usually focuses on the user activities, recording user actions through the Visual AAT when initially creating Agendas and recreating those actions during subsequent test sessions. The focus on user activities represents a high-level, conceptual approach to test session design.

Sometimes a tester may prefer to use a low-level, "nuts-and-bolts" approach that focuses on specific internal implementation commands, such as HTTP transactions. The WebLOAD DOM extension set includes objects, methods, properties, and functions that support this approach. Items in the *WebLOAD JavaScript Reference Manual* that are relevant to the HTTP Transaction Mode, as opposed to the more commonly used User Activity Mode, are noted as such in the *Reference Manual* entries.

1.3 When would I edit the JavaScript in my Agendas?

The Visual AAT automatically creates JavaScript Agendas for test sessions based on the actions performed by the user during recording. You don't have to be familiar with the JavaScript language to work with WebLOAD and test Web applications. However, as your testing needs increase, you may want to edit and expand the set of Agendas that were already recorded. Many users prefer to design test sessions around a set of basic Agendas created through the Visual AAT and then expand or tailor those Agendas to meet a particular testing need. Some of the reasons for editing JavaScript Agendas include:

- ◆ Recycling and updating a useful library of test Agendas from earlier versions of WebLOAD or WebFT.
- ◆ Creating advanced, specialized verification functions.
- ◆ Debugging the application being tested.
- ◆ Optimization capabilities, to maximize your application's functionality at minimal cost.

This manual documents the syntax and usage of the actions, functions, objects, and variables provided by WebLOAD to add advanced functionality and tailoring to the JavaScript Agendas created through the Visual AAT. JavaScript is very similar to other object-oriented programming languages such as C++, Java, and Visual Basic. The syntax of JavaScript is also very similar to C. If you know any of these other languages, you will find JavaScript very easy to learn. You can probably learn enough about JavaScript to start programming just by studying the examples in this book.

Note: For detailed information about the JavaScript language, please refer to the section entitled *The Core JavaScript Language* in the *Netscape JavaScript Guide*, which is supplied in Adobe Acrobat format with the WebLOAD software. You may also learn the elements of JavaScript programming from many books on Web publishing. Keep in mind that some specific JavaScript objects relating to Web publishing do not exist in the WebLOAD test environment.

1.4 Accessing Agenda components

WebLOAD uses test session Agendas to simulate user activities at a Web site. An Agenda is initially created by the Visual AAT during a recording session. As a user works with a test application in a browser, (for example, navigating between pages, typing text into a form, or clicking on the mouse), the Visual AAT stores information about all user actions in an Agenda. Agendas are also edited through the Visual AAT. Users may add functionality or customize their Agendas through the objects, functions, and other features described in this manual.

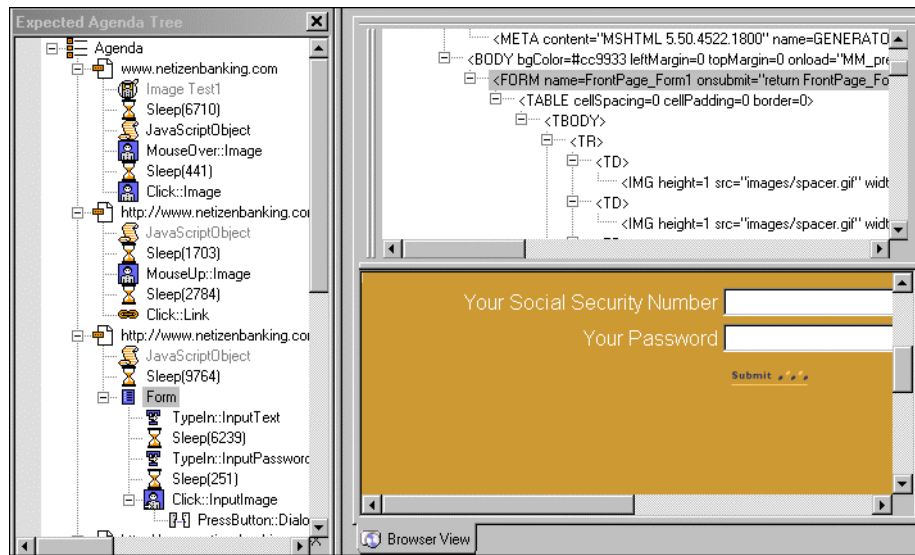
Customizing Agendas may involve nothing more than dragging an icon from the Visual AAT toolbar and dropping it into a graphic representation of the Agenda. It may involve entering or changing data through a user-friendly dialog box, or with the help of a Wizard. Some users may even add special features to their Agendas by editing the underlying code of the Agenda itself. When working with Agendas, users may be working on many different levels. For that reason, the Visual AAT desktop includes multiple view options, providing information on multiple levels. See Chapter 2, *Working with the Visual AAT*, in the *WebLOAD Programming Guide*, for a more extensive, illustrated explanation of the Visual AAT desktop components.

- ◆ Most users access Agendas primarily through an *Agenda Tree*, a set of clear, intuitive icons and visual devices representing user activities during a recording session, arranged into a logical structure. Each user activity in the Agenda Tree is referred to as a node. Nodes are organized in a hierarchical arrangement. The outmost level, or *root* level, is a single Agenda node. The second level directly under the root Agenda node includes all the Web pages to which the user navigated over the course of the recording session. The third level, organized under each Web page, includes all the user activities that occurred on the parent Web page. These activities are themselves organized into additional levels. For example, all data input on a single form in a Web page is organized into a single sub-tree of user input nodes collected under the node for that form. The Agenda Tree appears on the left side of the Visual AAT desktop.
- ◆ Web page nodes are added to the Agenda Tree in one of two ways. Some Web pages are the result of a user action on the previous page, such as clicking on a link and jumping to a new page. Other Web pages are created as a result of direct or indirect navigation, such as entering a URL in the browser window, or pop-up windows triggered by a previous navigation. The sets of user activities contained between two direct-navigation Web pages in an Agenda Tree parallel the *navigation blocks* found within the JavaScript Agenda code.

During a Visual AAT recording session, a new navigation block is created each time a user completes a direct navigation, manually entering a new URL into the Visual AAT address bar. Each navigation block is surrounded by a `try{} catch{}` statement in the corresponding JavaScript Agenda code. Navigation blocks are useful for error management, especially when running “hands-free” test sessions. For example, the user can define the default testing behavior to be that if an error is encountered during a test session, WebLOAD should throw the error, skip to the next navigation block, and continue with the test session. Errors during playback are indicated by a red X appearing beside the problematic action in the Agenda Tree.

- ◆ The user activity icons, or nodes, in an Agenda Tree have certain characteristics, or properties, associated with them. For example, a message icon would be associated with a message text and some severity level. The properties associated with a selected node are displayed in a list in the *Properties pane*. To edit unprotected property values, select a property field and either enter new data or choose from the alternate values provided in a convenient drop-down list. The Properties pane appears on the right side of the Visual AAT desktop.
- ◆ The graphic nodes in an Agenda Tree actually represent blocks of code within the underlying recorded Agenda. The JavaScript code corresponding to a selected node is automatically displayed in the *JavaScript View pane*. The JavaScript View pane is one of the tabs available in the center of the Visual AAT desktop.
- ◆ The graphic nodes in an Agenda Tree represent user actions on a Web site. An exact replica, or snapshot, of each user activity is stored during recording and available in the Visual AAT Browser View to aid in debugging and help users remember what each action accomplished. The *Browser View pane* is one of the tabs available in the center of the Visual AAT desktop.
- ◆ Web pages are created through HTML programs. The HTML code that underlies each stored Web page is also stored during recording sessions. For easy reference, the HTML code of the Web page associated with a selected node is displayed in the HTML View pane. The *HTML View pane* is one of the tabs available in the center of the Visual AAT desktop.
- ◆ Web pages have a logical structure that may be represented through a series of DOM object trees. The DOM tree for a selected Web page is essentially a hierarchically structured, more easily understood representation of the DOM objects found in the HTML code for that Web page. The DOM tree of the Web page associated with a selected node is displayed in the DOM View pane. The *DOM View pane* is one of the tabs available in the center of the Visual AAT desktop. When working with the DOM View, the center pane is actually split in half, with the upper half displaying the DOM View and the lower half displaying the corresponding Web page, seen in the Browser View.

The following figure illustrates a Visual AAT desktop displaying the Agenda Tree and DOM and Browser Views for the highlighted Form node. The Agenda Tree is on the left with the icon representing a form field on a Web page highlighted. The actions taken by the user while working with the selected form all appear in the Agenda Tree as a single branch of the Agenda Tree, under the selected form. The Browser View pane on the lower right focuses on a piece of the selected form as it appeared on the Web page at the time this Agenda was recorded. The DOM View pane on the upper right displays the DOM objects that represent the selected form, arranged in a tree that corresponds to the user activity in the selected form.



Introduction to JavaScript Agendas

Figure 1-1: Visual AAT desktop: Agenda Tree, DOM View, and Browser View panes for selected Form node branch

1.5 Editing the JavaScript code in an Agenda

1.5.1 Accessing the JavaScript code within the Agenda Tree

The Visual AAT provides a complete graphic user interface for creating and editing Agenda files. Additions or changes to an Agenda are usually made through the Visual AAT GUI, working with intuitive icons representing user actions in a graphic Agenda Tree. For greater clarity, the JavaScript code that corresponds to each user action in an Agenda is also visible in the JavaScript View pane on the Visual AAT desktop.

While most people never really work with the JavaScript code within their Agenda, some users do wish to manually edit the JavaScript code underlying their Agenda Tree. For example, some test sessions may involve advanced WebLOAD testing features that can not be completely implemented through the GUI, such as Java or XML objects. Editing the JavaScript code in an Agenda does not necessarily mean editing a huge JavaScript file. Most of the time users only wish to add or edit a specific feature or a small section of the code. The Visual AAT provides access to the JavaScript code in an Agenda through JavaScript Object nodes, which are seen on the following levels:

- ◆ JavaScript Object nodes—individual nodes in the Agenda Tree. Empty JavaScript Object nodes may be dragged from the Visual AAT toolbar and dropped onto the Agenda Tree at any point selected by the user, as described in *Adding JavaScript Object Nodes*, page 75, in the *WebLOAD Programming Guide*. Use the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 17, to add lines of code or functions to the JavaScript Object.
- ◆ Converted Web page—the sub-tree or branch of an Agenda Tree that represents all user activity within a single Web page, converted to a single JavaScript Object node. A Web page branch is ‘rooted’ in the Agenda Tree with an icon that represents the user’s navigation to that page’s URL. The icons on that branch represent all user activities from the point at which that Web page was first accessed until the point at which the user navigated to a different Web page. Some testing features may require manually editing or rewriting the JavaScript code for user activities within a Web page. To manually edit code in a recorded Agenda, the Web page branch that includes that code must be converted to a JavaScript Object. Converting a Web page branch to a JavaScript Object is simple. Right click on the preferred Web page node in the Agenda Tree and select Convert to JavaScript Object from the pop-up menu. The entire Web page branch becomes a single JavaScript Object, which can then be edited through the IntelliSense Editor. Note that once a branch has been converted to a single JavaScript Object, the various user activity icons that were on that branch are no longer individually accessible.
- ◆ Imported JavaScript File—an external JavaScript file that should be incorporated within the body of the current Agenda. Select Edit | Import JavaScript File from the Visual AAT menu to import the file of your choice. Often testers work with a library of pre-existing library files from which they may choose functions that are relevant to the current test session. This modular approach to programming simplifies and speeds up the testing process, and is fully supported and endorsed by WebLOAD.

- ◆ Converted Agenda Tree—if necessary, an entire Agenda Tree can be converted to a single JavaScript Object node consisting of a straight JavaScript text file. Right click on the Agenda Tree's root node and choose Convert to JavaScript Object from the pop-up menu. However, this conversion is not recommended unless manual editing of an entire Agenda file is truly required for the test session.

1.5.2 Using the IntelliSense JavaScript Editor

For those users who wish to manually edit their Agendas, the Visual AAT provides three levels of programming assistance:

- ◆ An IntelliSense Editor mode for the JavaScript View pane.

Add new lines of code to your Agenda or edit existing JavaScript functions through the IntelliSense Editor mode of the JavaScript View pane. The IntelliSense Editor helps programmers write the JavaScript code for a new function by formatting new code and prompting with suggestions and descriptions of appropriate code choices and syntax as programs are being written. For example, in the following figure the IntelliSense Editor displays a drop-down list of available properties and objects for the `wlHttp` object being added to the program, with a pop-up box describing the highlighted method in the list.

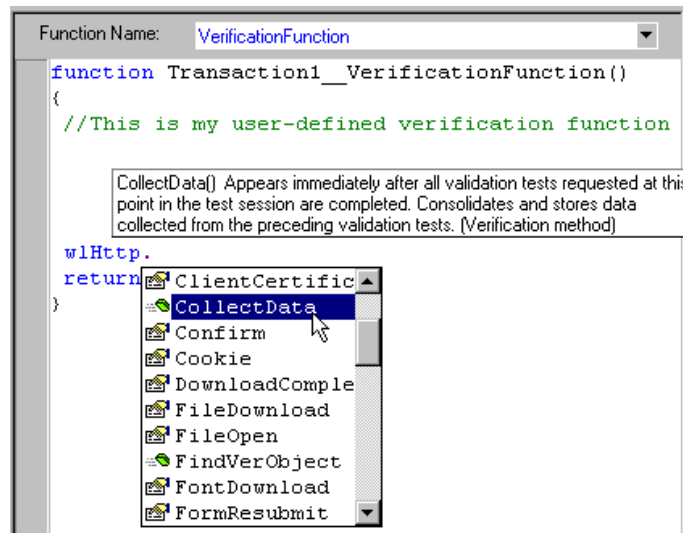


Figure 1-2: IntelliSense Editor mode

- ◆ A selection of the most commonly used programming constructs, available through the Edit | Insert | JavaScript menu.

Users who choose to program their own JavaScript Object code within their Agenda may still take advantage of the Visual AAT GUI to simplify their programming efforts. Rather than manually type out the code for each command, with the risk of making a mistake, even a trivial typo, and adding invalid code to the Agenda file, users may select an item from the Edit | Insert | JavaScript menu, illustrated in the following figure, to bring up a list of

available commands and functions for the selected item. The Visual AAT automatically inserts the correct code for the selected item into the JavaScript Object currently being edited. The user may then change specific parameter values without any worries about accidental mistakes in the function syntax.

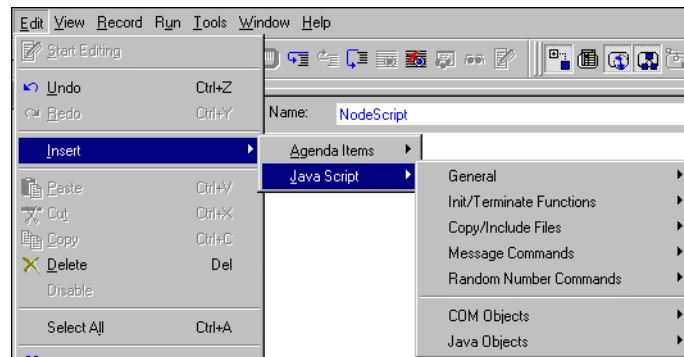


Figure 1-3: Automatic JavaScript command list

- ◆ A Syntax Checker that checks the syntax of the code in your Agenda file and catches simple syntax errors before you spend any time running a test session. Select Tools | CheckSyntax while standing in the JavaScript View pane of the Visual AAT desktop to check the syntax of the code in your Agenda file.

Agenda code that you wish to write or edit must be part of a JavaScript Object in the Agenda Tree. Adding or converting JavaScript Objects in an Agenda Tree is described in *Accessing the JavaScript code within the Agenda Tree*, page 16.

Note: If you do decide to edit the JavaScript code in an Agenda, note the following points:

- Be careful not to damage the Agenda structure by changing the sequence or integrity of the Agenda navigation blocks. A test session Agenda is constructed and based on a specific sequence of user activities, such as Web page navigations, mouse clicks, form submissions, and links. Changing the sequence of code blocks in effect means changing the sequence of user activities, and may destroy the functionality of the test session Agenda.
- If you use an external text editor to modify the JavaScript code in a JavaScript Agenda that was created through the Visual AAT, the changes you made through the external editor will be lost if you open the Agenda in the Visual AAT again. Therefore, be sure to do all JavaScript code editing through the IntelliSense Editor in the Visual AAT only.
- For detailed information about the JavaScript language, please refer to the section entitled *The Core JavaScript Language* in the *Netscape JavaScript Guide*. This manual is supplied in Adobe Acrobat format with the WebLOAD software. You may also learn the elements of JavaScript programming from many books on Web publishing. Keep in mind that some specific JavaScript objects relating to Web publishing do not exist in the WebLOAD test environment.

WebLOAD Actions, Objects, and Functions

action (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

form

Description

Specifies the URL to which the form contents are to be sent for processing (read-only string).

Syntax

<NA>

Example

Based on the following <FORM> tag:

```
<FORM name="SignUp"
  action="http://www.ABCDEF.com/FormProcessor.exe"
  method="post">
```

The action is:

```
"http://www.ABCDEF.com/FormProcessor.exe"
```

See also

form, page 102

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Actions

Description

WebLOAD runs test Agendas that simulate the actions of real users. As you complete actions in the browser, (for example, mouse clicks, link clicks, or form entries), the Visual AAT records your actions in a JavaScript Agenda. These actions are then recreated during subsequent test sessions. User actions are therefore one of the basic components of JavaScript Agendas.

This manual provides detailed reference information on the WebLOAD set of user actions. These actions are accessed within JavaScript Agendas as methods of various WebLOAD objects. The objects themselves are documented here as well. All actions and objects are listed alphabetically.

Syntax

```
Object.ActionMethod()
```

Example

Each individual method includes a syntax specification and example for that method.

See also

AutoNavigate(), page 26

event, page 91

Navigate(), page 186

OnMouseOver, page 191

SetWindow(), page 244

wlClick(), page 326

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

wlTypeIn(), page 366

Back(), page 27

Forward(), page 106

OnClick, page 190

Refresh(), page 213

wlBrowser, page 325

wlClose(), page 329

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlSubmit(), page 362

Add() (method)

Method of Objects

wlGeneratorGlobal

wlSystemGlobal

Description

Adds the specified number value to the specified shared integer variable.

Syntax

```
Add("SharedIntVarName", number, ScopeFlag)
```

Parameters

SharedIntVarName—The name of a shared integer variable to be incremented.

number—An integer with the amount to add to the specified variable.

ScopeFlag—One of two flags, *WLCurentAgenda* or *WLAllAgendas*, signifying the scope of the shared variable.

When used as a method of the *wlGeneratorGlobal* object:

- ♦ The *WLCurentAgenda* scope flag signifies variable values that you wish to share between all threads of a single Agenda, part of a single process, running on a single Load Generator.
- ♦ The *WLAllAgendas* scope flag signifies variable values that you wish to share between all threads of one or more Agendas, common to a single spawned process, running on a single Load Generator.

When used as a method of the *wlSystemGlobal* object:

- ♦ The *WLCurentAgenda* scope flag signifies variable values that you wish to share between all threads of a single Agenda, potentially shared by multiple processes, running on multiple Load Generators, system wide.
- ♦ The *WLAllAgendas* scope flag signifies variable values that you wish to share between all threads of all Agendas, run by all processes, on all Load Generators, system-wide.

Return Value

None

Example

```
wlGeneratorGlobal.Add("MySharedCounter", 5, WLCurentAgenda)  
wlSystemGlobal.Add("MyGlobalCounter", 5, WLCurentAgenda)
```

See also*Get()*, page 110*Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide**Set()*, page 237*wlGeneratorGlobal*, page 334*wlSystemGlobal*, page 363

addProperty() (method)

Method of Object*WSComplexObject***Description**

Defines and adds properties to a new `WSComplexObject` object. The new properties must correspond to the properties defined for the object in a WSDL file.

Syntax

```
newComplexObject.addProperty("PropertyName", "PropertyType")
```

Parameters

`PropertyName`—Name of the property being defined, a text string.

`PropertyType`—Type of the property being defined, such as “integer” or “string”, a text string.

Return Value

None.

Example

```
Worker = new WSComplexObject();
Worker.setType("Worker");
Worker.addProperty("Name", "string");
Worker.addProperty("Age", "integer");
```

Comment

New `WSComplexObject` objects must be created, and their properties defined, within the `InitClient()` function. Otherwise, a new object will be created with each iteration of the Agenda during a test session and the system will quickly run out of memory.

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

setType(), page 242

setValue(), page 243

WSComplexObject, page 374

WSComplexObject(), page 375

WSGetSimpleValue(), page 376

WSWebService, page 377

WSWebService(), page 379

AdjacentText (property)

Property of Object

element

Checkbox

InputCheckbox

InputRadio

Radiobutton

Description

The text that appears near a Checkbox or Radiobutton.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

element, page 83

Checkbox, page 38

InputCheckbox, page 158

InputRadio, page 162

Radiobutton, page 210

Alt (property)

Property of Object

area

element

Image

InputImage

Description

The text displayed within the area, input, or image object. When working with image elements, this text is the alternative descriptive string to be displayed if the current image can not be displayed. Otherwise functions as a tooltip.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

area, page 24

element, page 83

Image, page 148

InputImage, page 160

area (object)

Description

Each area object defines the shape, coordinates, and associated URL of one hyperlink region within a client-side image map. area objects are accessed through collections of `links` or `map.areas`.

Syntax

<NA>

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

Properties

Alt, page 23

ContainingMap, page 55

Coords, page 57

id, page 143

Shape, page 246

Url, page 302

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also*Collections*, page 47*document*, page 79*link*, page 169*map*, page 178

Automatic State Management for HTTP Protocol Mode

Mode

The ASM components listed here are usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties and Methods of Object*wlHttp***Description**

WebLOAD provides a high-level working environment for users, recording any objects of interest to users without the user needing to be aware of the objects' underlying DOM properties or identification details. During subsequent playback test sessions, some of the recorded objects' properties may change slightly, due to the dynamic nature of Web activity. For example, an object's location or URL may be different from that originally recorded. WebLOAD provides full access to the complete set of objects originally of interest to the users at recording time, even if they include a slightly altered set of DOM properties. WebLOAD automatically incorporates the *Dynamic Object Recognition (DOR) components*, described on page 81, to compensate for the dynamic nature of most Web sites. Users who work in the HTTP Protocol Mode use a slightly different approach to dynamic object recognition, based on the properties and methods listed in this section.

Properties and Methods

The ASM property set includes the following properties and methods:

ExpectedDocument, page 93*ExpectedDOMID*, page 94*ExpectedID*, page 94*ExpectedLocation*, page 95*ExpectedName*, page 96*ExpectedText*, page 97*GetCurrentValue()*, page 114*type*, page 300*wlTarget*, page 366**Example**

```
wlHttp.ExpectedID = ""
wlHttp.ExpectedName = ""
wlHttp.ExpectedText = "Pisces"
```

```

wlHttp.Type = "A"
wlHttp.ExpectedLocation = ":#1.#1:"
    // WebLOAD shorthand notation, described in
    // Identifying frame locations, page 224
    // in the WebLOAD Programming Guide

wlHttp.Url =
    wlHttp.IdentifyObject("http://www.pisces.com/info/index.cfm?
    act=0&PID=1425&Parent=10&CFID=70466&CFTOKEN=61110909")

wlHttp.Get(wlHttp.Url)

```

Comment

Agendas that work within the User Activity mode use a slightly different approach to dynamic object recognition. See *Dynamic Object Recognition (DOR) components*, page 81, for more information.

See also

Dynamic Object Recognition (DOR) components,
page 81

wlHttp, page 342

Identifying frame locations, page 224 in the
WebLOAD Programming Guide

Working in HTTP Protocol Mode, page 213 in the
WebLOAD Programming Guide

AutoNavigate() (action)

Method of Object

wlBrowser

Description

Automatic browser activity triggered by an automatic navigation request, such as an automatic popup window for advertisements sent by the server, or if a page includes code that calls for an automatic refresh every 10 seconds.

Syntax

```
wlBrowser.AutoNavigate()
```

Parameters

None

Return Value

None

Example

<NA>

See also*Actions*, page 20*Back()*, page 27*Forward()*, page 106*OnClick*, page 190*Refresh()*, page 213*wlBrowser*, page 325*wlClose()*, page 329*wlMouseOver()*, page 348*wlMultiSelect()*, page 350*wlSubmit()*, page 362*AutoNavigate()*, page 26*event*, page 91*Navigate()*, page 186*OnMouseOver*, page 191*SetWindow()*, page 244*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseUp()*, page 349*wlSelect()*, page 358*wlTypeIn()*, page 366

Back() (action)

Method of Object*wlBrowser***Description**

Get and display the previous Web page.

Syntax`wlBrowser.Back()`**Parameters**

None.

Return Value

None

Example

<NA>

Comment

This method implements a special category of user action—clicking on a specific browser shortcut button from the Microsoft IE toolbar.

See also*Actions*, page 20*AutoNavigate()*, page 26

<i>Back()</i> , page 27	<i>event</i> , page 91
<i>Forward()</i> , page 106	<i>Navigate()</i> , page 186
<i>OnClick</i> , page 190	<i>OnMouseOver</i> , page 191
<i>Refresh()</i> , page 213	<i>SetWindow()</i> , page 244
<i>wlBrowser</i> , page 325	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlSubmit()</i> , page 362	<i>wlTypeIn()</i> , page 366

BeginTransaction() (function)

Mode

This function is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

Use the `BeginTransaction()` and `EndTransaction()` functions to define the start and finish of a logical block of code that you wish to redefine as a single logical transaction unit. This enables setting timers, verification tests, and other measurements for this single logical unit.

Syntax

```
BeginTransaction(TransName)
...
    <any valid JavaScript code>
...
EndTransaction(TransName, Verification, [SaveFlag])
```

Parameters

`TransName`—The name assigned to this transaction, a user-supplied string.

Return Value

None

Example

<NA>

GUI mode

Note that `BeginTransaction()` and `EndTransaction()` functions are usually accessed and inserted into Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates a Form branch in the Agenda Tree bracketed by `BeginTransaction` and `EndTransaction` nodes. The `EndTransaction` node is highlighted in the Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right. See *Adding transactions*, page 49 in the *WebLOAD Programming Guide*, for more information.

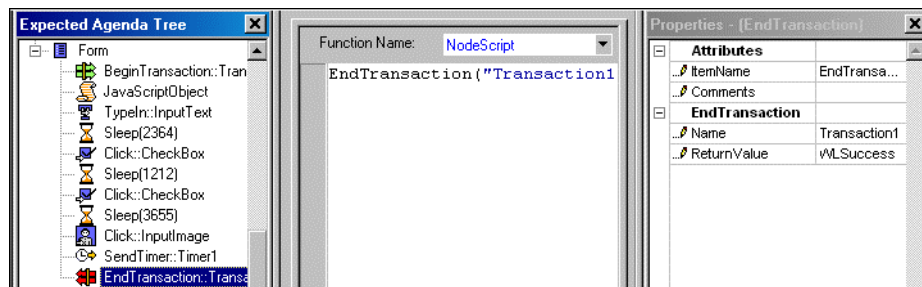


Figure 2-1: EndTransaction added to an Agenda

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

bitrate (property)

Property of Object

wlMediaPlayer()

Description

Retrieves a value indicating the bit rate for the stream in bits per second. (read-only long number)

Syntax

`MyMediaPlayerObject.bitrate`

Example

<NA>

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

Browser configuration components

Mode

Many of the following properties and methods are usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties and Methods of Objects

wlGlobals

wlHttp

wlLocals

Description

The *wlGlobals*, *wlLocals*, and *wlHttp* objects share a set of components that manage user browser activities. This section lists these browser properties and methods. Some of the

components are common to all three objects. Some of the properties or methods are used by only one object, and are marked so in the tables. Most of the properties listed here are used only if working in HTTP Protocol mode, described in *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*.

Note: The values assigned in a `wlHttp` object override any global defaults assigned in `wlGlobals` or local defaults in `wlLocals`. WebLOAD uses the `wlGlobals` or `wlLocals` defaults only if you do not assign values to the corresponding properties in the `wlHttp` object. For more information about working with both global and local values, see *Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide*.

Syntax

```
NewValue = wlGlobals.BrowserMethod()
wlGlobals.BrowserProperty = PropertyValue
```

Example

Each individual property and method includes examples of the syntax for that property.

Methods

ClearDNSCache(), page 41

ClearSSLCache(), page 42

The following methods are for `wlHttp` objects only

CloseConnection(), page 46

Get(), page 111

IdentifyObject(), page 147

Post(), page 202

Data Methods

Head(), page 137

wlClear(), page 328

wlGet(), page 335

wlSet(), page 359

Properties

The following properties are for `wlHttp` objects only

Data Properties

Data, page 63

DataFile, page 65

Erase, page 87

fileName, page 100

FormData, page 104

Header, page 138

DataCollection.type, page 300

DataCollection.value, page 310

ASM Properties

ExpectedDocument, page 93

ExpectedDOMID, page 94

ExpectedID, page 94

ExpectedLocation, page 95

ExpectedName, page 96

ExpectedText, page 97

type, page 300

wlTarget, page 366

The following properties are used by *wlHttp*, *wlLocals*, and *wlGlobals* objects unless otherwise noted.

Configuration Properties

BrowserCache, page 33

ConnectionSpeed, page 54 (*wlGlobals* only)

DefaultAuthentication, page 66

DisableSleep, page 75

DNSUseCache, page 78

KeepAlive, page 165

LoadGeneratorThreads, page 173

MultiIPSupport, page 183

NTUserName, *NTPassWord*, page 187

Outfile, page 197

Password, page 198

ProbingClientThreads, page 206

Proxy, *ProxyUserName*, *ProxyPassWord*, page 209

RedirectionLimit, page 212

SaveSource, page 226

SaveTransaction, page 227 (*wlGlobals* only)

SaveWSTransaction, page 228
(*wlGlobals* only)

SSLBitLimit, page 257 (*wlGlobals* only)

SSLCryptoStrength, page 262
(*wlGlobals* only)

SSLClientCertificateFile,
SSLClientCertificatePassword, page 260

SSLUseCache, page 273

SSLVersion, page 275

Url, page 302

UserAgent, page 306

UserName, page 307

UsingTimer, page 308

Version, page 322

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*.

BrowserCache (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects*wlGlobals**wlHttp**wlLocals***Description**

Enable simulation of the typical browser cache mode. The possible values of `BrowserCache` are:

- ◆ No—Disable browser caching.
- ◆ Yes—Enable browser caching. (default)

A "Yes" value means that WebLOAD will store the most recent downloaded images in a virtual browser cache. This speeds up subsequent document accesses, just as using a real browser cache speeds up access time in real use. The browser cache is automatically cleared at the end of each round.

GUI mode

WebLOAD recommends enabling or disabling the browser cache through the Console GUI. Enable caching for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

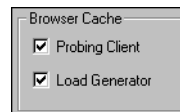


Figure 2-2: Enabling Browser Cache for Load Generator

Syntax

<NA>

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Button (object)

Property of Objects

`Button` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

When working with a document object, each `Button` object represents one of the buttons embedded in a document, that is, specifies a container for rich HTML that is rendered as a button. When working with a mouse object, used to set or retrieve the mouse button pressed by the user. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["BUTTON"]
```

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

event, page 91

InnerText, page 155

OnClick, page 190

value, page 310

id, page 143

Name, page 184

title, page 296

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputRadio, page 162*InputText*, page 164*length*, page 167*Radiobutton*, page 210*Select*, page 231*text*, page 291*TextArea*, page 292*UIContainer*, page 301

cell (object)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

`cell` objects are grouped into collections of `cells`. The `cells` collection is a property of the following objects:

*row**wlTable*

Description

A `cell` object contains all the data found in a single table cell. If the `cells` collection is a property of a `wlTables` object, then the collection refers to all the cells in a particular table. If the `cells` collection is a property of a `row` object, then the collection refers to all the cells in a particular row. Individual `cell` objects may be addressed by index number, similar to any object within a collection.

Syntax

Individual `cell` objects may be addressed by index number, similar to any object within a collection. For example, to access a property of the 16th cell in `myTable`, counting across rows and with the first cell indexed at 0, you could write:

```
document.wlTables.myTable.cells[15].<cell-property>
```

If you are working directly with the cells in a `wlTable` object, as opposed to the cells within a single `row` object, you may also specify a range of cells from anywhere within the table using the standard spreadsheet format. Specify a group of cells using a string with the following format:

- ◆ Use *letters* to indicate columns, starting with the letter **A** to represent the first column.
- ◆ Use *numbers* to indicate rows, starting with the number **1** to represent the first column. (Note that this is not typical—the standard JavaScript index begins at **0** to represent the first element in a set.)

Example**For cells within a w1Tables object:**

```
document.w1Tables.myTable.cells["A1:C3"]
```

In this example, the string "A1:C3" includes all cells from the first column of the first row up to the third column in the third row, *reading across rows*. This means that the first cell read is in the first column of the first row, the second cell read is in the *second column* of the *first row*, the third cell read is in the third column of the first row, and so on until the end of the first row. If the table includes eight columns, then the ninth cell read will be in the first column of the second row, and so on.

For cells within a row object:

To access a property of the 4th cell in the 3rd row in `myTable`, counting across rows and with the first cell indexed at 0, you could write:

```
document.w1Tables.myTable.rows[2].cells[3].<cell-property>
```

Note that individual table cells often are merged and span multiple rows. In such a case, the cell will only appear in the collection for the *first* of the set of rows that the cell spans.

Properties

Each `cell` object contains information about the data found in one cell of a table. The `cell` object includes the following properties:

cellIndex, page 37

InnerHTML, page 153

InnerText, page 155

tagName, page 289

Comment

`cell` is often accessed through the `w1Tables` family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.w1Tables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

cell, page 35 (`w1Table` and `row` property)

cellIndex, page 37 (`cell` property)

Collections, page 47

cols, page 48 (`w1Table` property)

Compare(), page 49

CompareColumns, page 51

CompareRows, page 52

Details, page 72

<i>id</i> , page 143 (<i>wlTable</i> property)	<i>InnerHTML</i> , page 153 (<i>cell</i> property)
<i>InnerText</i> , page 155 (<i>cell</i> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<i>wlTable</i> property)	<i>rowIndex</i> , page 225 (<i>row</i> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<i>cell</i> property)
<i>wlTable</i> , page 364	
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

cellIndex (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

cell

Description

An integer containing the ordinal index number of this *cell* object within the parent table or row. Cells are indexed starting from zero, so the *cellIndex* of the first cell in a table or row is 0.

Syntax

<NA>

Comment

cellIndex is a member of the *wlTables* family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard *document.all* collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

<i>cell</i> , page 35 (<i>wlTable</i> and <i>row</i> property)	<i>cellIndex</i> , page 37 (<i>cell</i> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<i>wlTable</i> property)

<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<i>wlTable</i> property)	<i>InnerHTML</i> , page 153 (<i>cell</i> property)
<i>InnerText</i> , page 155 (<i>cell</i> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<i>wlTable</i> property)	<i>rowIndex</i> , page 225 (<i>row</i> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<i>cell</i> property)
<i>wlTable</i> , page 364	
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

Checkbox (object)

Property of Objects

Checkbox objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Each `Checkbox` object represents one of the checkboxes embedded in a document. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["CHECKBOX"]
```

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358

Properties

<i>AdjacentText</i> , page 23	<i>id</i> , page 143
<i>Name</i> , page 184	<i>title</i> , page 296
<i>value</i> , page 310	

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

checked (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

element

Description

For an `<INPUT type="checkbox">` or `<INPUT type="radio">` element, the `checked` property indicates whether the element has the HTML `checked` attribute, that is, whether the element is selected. The property has a value of `true` if the element has the `checked` attribute, or `false` otherwise (read-only).

Syntax

`<NA>`

See also

element, page 83

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ClearAll() (method)

Method of Object

wlCookie

Description

Delete all cookies set by *wlCookie* in the current thread.

Syntax

```
wlCookie.ClearAll()
```

Parameters

None

Return Value

None

Example

<NA>

See also

wlCookie, page 330

ClearDNSCache() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects

wlGlobals

wlHttp

wlLocals

Description

Clear the IP address cache.

Syntax

```
wlHttp.ClearDNSCache()
```

Parameters

None

Return Value

None

Example

<NA>

GUI mode

WebLOAD recommends enabling or disabling the DNS cache through the Console GUI. Disable caching for the Load Generator or for the Probing Client during a test session by clearing the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

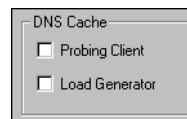


Figure 2-3: Clearing DNS Cache for Load Generator

Comment

To enable or disable DNS caching when working in HTTP Protocol mode, set the *DNSUseCache* property.

See also

Browser configuration components, page 30

ClearSSLCache(), page 42

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlLocals, page 343

ClearDNSCache(), page 41

DNSUseCache, page 78

SSLUseCache, page 273

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ClearSSLCache() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects*wlGlobals**wlHttp**wlLocals***Description**

Clear the SSL decoding-key cache.

Syntax

```
wlHttp.ClearSSLCache()
```

Parameters

None

Return Value

None

Example

<NA>

GUI mode

WebLOAD recommends enabling or disabling the SSL cache through the Console GUI. Disable caching for the Load Generator or for the Probing Client during a test session by clearing the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

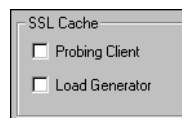


Figure 2-4: Clearing SSL Cache for Load Generator

Comment

To enable or disable SSL caching when working in HTTP protocol mode, set the *SSLUseCache* property.

See also

Browser configuration components, page 30

ClearDNSCache(), page 41

ClearSSLCache(), page 42

DNSUseCache, page 78

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

SSLUseCache, page 273

wlGlobals, page 339

wlHttp, page 342

wLocals, page 343*Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*

ClientNum (variable)

Description

`ClientNum` is set to the serial number of the client in the WebLOAD test configuration. `ClientNum` is a read-only local variable. Each client in a Load Generator has a unique `ClientNum`. However, two clients in two different Load Generators may have the same `ClientNum`. Note that while `ClientNum` is unique within a single Load Generator, it is not unique system wide. Use `VCUniqueID()` to obtain an ID number which is unique system-wide.

If there are N clients in a Load Generator, the clients are numbered $0, 1, 2, \dots, N-1$. You can access `ClientNum` anywhere in the local context of the Agenda (`InitClient()`, main script, `TerminateClient()`, etc.). `ClientNum` does not exist in the global context (`InitAgenda()`, `TerminateAgenda()`, etc.).

If you mix Agendas within a single Load Generator, instances of two or more Agendas may run simultaneously on each client. Instances on the same client have the same `ClientNum` value.

`ClientNum` reports only the main client number. It does not report any extra threads spawned by a client to download nested images and frames (see *LoadGeneratorThreads*, page 173).

Example

<NA>

Comment

Earlier versions of WebLOAD referred to this value as `ThreadNum`. The variable name `ThreadNum` will still be recognized for backward compatibility.

GUI mode

WebLOAD recommends accessing global system variables, including the `ClientNum` identification variable, through the Visual AAT GUI. To see a list of global system variables, highlight the top-level Agenda node in the Agenda Tree, find the `GlobalVariables` section of the Properties pane, and click on the browse button of the System field. A complete list of system variables and macros pops up, as illustrated in the following figure. The variables that appear in this list are available for use at any point in an Agenda file. For example, it is convenient to add `ClientNum` to a Message Node to clarify which client sent the messages that appear in the Console Log window.

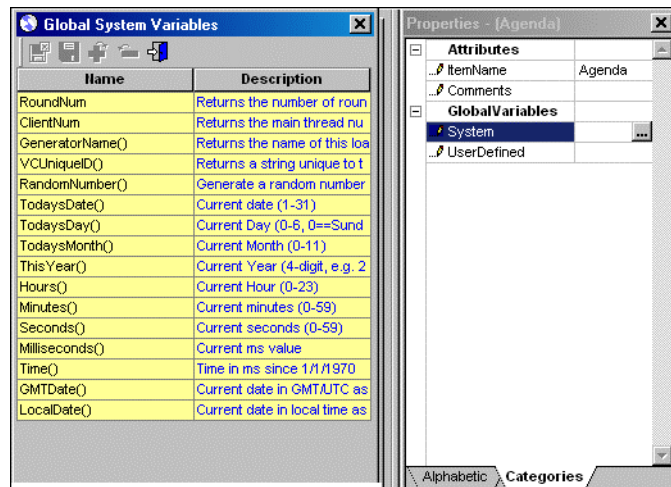


Figure 2-5: Global System Variables List

See also*ClientNum*, page 43*GeneratorName()*, page 108*GetOperatingSystem()*, page 130*Identification variables and functions*, page 146*RoundNum*, page 221*VCUniqueID()*, page 312

Close() (function)

Method of Object*wlOutputFile***Description**

Closes an open file. When called as a method of the `wlOutputFile` object, closes the open output file being managed by that object.

Syntax**Function call:**

```
Close(filename)
```

wlOutputFile method:

```
wlOutputFile.Close()
```

Parameters**Function call:**

`filename`—A string with the name of the ASCII output file to be closed.

wlOutputFile method:

No parameter is necessary when this function is called as a method of the `wlOutputFile` object, since the file to be closed is already known.

Return Value

None.

Example

Function call:

```
Close(MyFavoriteFile)
```

wlOutputFile method:

```
MyFileObj = new wlOutputFile(filename)
...
MyFileObj.Close()
```

Comment

When you use the `Close()` function to close a file, data will be flashed to the disk.

See also

Close(), page 45

delete(), page 70

GetLine(), page 124

Open(), page 193

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

wlOutputFile, page 351

Write(), page 372

CopyFile(), page 57

File management functions, page 100

IncludeFile(), page 150

Reset(), page 219

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile(), page 354

WriteLn(), page 373

CloseConnection() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHttp

Description

Closes an HTTP connection that was opened with the *KeepAlive* option.

Syntax

```
wlHttp.CloseConnection()
```

Parameters

None

Return Value

None

Example

<NA>

GUI mode

WebLOAD recommends maintaining or closing connections through the Console GUI. Enable maintaining connections for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

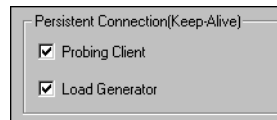


Figure 2-6: Enabling Persistent Connections for Load Generator

See also

Browser configuration components, page 30

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

KeepAlive, page 165

wlHttp, page 342

Collections

Description

Collections are arrays or sets of individual objects. For example, the `elements` collection refers to a collection of individual `element` objects.

Access individual members of a collection either through an index number or directly through the member's name or ID. The following three syntax choices are equivalent:

```
Collection[index#]
Collection["ID"]
Collection.ID
```

Test session Agendas work with all browser DOM collections and objects. The recommended way to access these objects is through the classic browser `document` object, via the `document.all` property, by checking the tag type. For example, access a table through:

```
document.all.tags["TABLE"]
```

Properties

Each collection of objects includes the single property `length`, which contains the size of the collection, that is, the number of objects included in this collection. You may also use the index value to access individual objects from within a collection.

For example, to find out how many `table` objects are contained within the `wlTables` collection of a document, check the value of:

```
document.wlTables.length
```

In this *Reference Manual*, the description of each individual object includes information on the collection, if any, to which that object belongs.

See also

element, page 83

length, page 167

cols (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

element

wlTable

Description

When working with `wlTables` objects, an integer containing the number of columns in this table. The column number is taken from the COLS attribute in the `<TABLE>` tag. This property is optional. If the table does not have a COLS attribute then the value is undefined. When working with `element` objects of type `TextArea`, an integer containing the number of columns in this `TextArea`.

Syntax

```
<NA>
```

Example

<NA>

Comment

`cols` is often accessed through the `wlTables` family of `table`, `row`, and `cell` objects. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

cell, page 35 (`wlTable` and `row` property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

element, page 83

InnerHTML, page 153 (`cell` property)

MatchBy, page 178

ReportUnexpectedRows, page 216

rowIndex, page 225 (`row` property)

tagName, page 289 (`cell` property)

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

cellIndex, page 37 (`cell` property)

cols, page 48 (`wlTable` property)

CompareColumns, page 51

Details, page 72

id, page 143 (`wlTable` property)

InnerText, page 155 (`cell` property)

Prepare(), page 205

row, page 223 (`wlTable` property)

TableCompare, page 286

wlTable, page 364

Compare() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

TableCompare

Description

The **recommended** usage of the `Compare` method compares the *expected* table specified when the `TableCompare` object was first created to the *actual* table specified here.

The **alternate** usage of the `Compare` method compares the expected and actual tables specified when the `TableCompare` object was first created.

Syntax**Recommended:**

```
Compare (Table)
```

Alternate:

```
Compare ()
```

Parameters**Recommended:**

`Table`—A pointer to the actual table to be compared by this method call. Recommended.

Alternate:

No parameter is necessary when working with the alternate usage of this method..

Return Value

`true` if the comparison succeeds.

`false` if the comparison fails. If the comparison fails, it shows up as a `TableCompareEvent` entry in the Data Drilling report, described in *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*.

Example**Recommended:**

```
RetVal = a.Compare (ActualTable)
```

Alternate:

```
RetVal = b.Compare ()
```

Comment

The comparison is completed using the settings defined by the user through the `CompareColumns` and `CompareRows` `TableCompare` properties.

The `Compare ()` method is used with the `TableCompare` Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

cell, page 35 (*wlTable* and *row* property)
Collections, page 47
Compare(), page 49
CompareRows, page 52
id, page 143 (*wlTable* property)
InnerText, page 155 (*cell* property)
Prepare(), page 205
row, page 223 (*wlTable* property)
TableCompare, page 286
wlTable, page 364
Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

cellIndex, page 37 (*cell* property)
cols, page 48 (*wlTable* property)
CompareColumns, page 51
Details, page 72
InnerHTML, page 153 (*cell* property)
MatchBy, page 178
ReportUnexpectedRows, page 216
rowIndex, page 225 (*row* property)
tagName, page 289 (*cell* property)

CompareColumns (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

TableCompare

Description

Specifies which columns to compare within the rows selected by the *MatchBy* property. Note that the *CompareRows* property must have been defined as "all".

WebLOAD offers the following options:

- Match all Columns
- Match Specific Columns

Syntax

String ["all"] (default)

or

Array of any combination of:

- Integer column numbers [*x*, *y*, *z*, etc.]

- String range of column numbers ["1-3", "6-8", etc.]
- String column names ["ColumnA", "ColumnB", etc.]

Example

```
a.CompareColumns = [1, 3, "7-13"]
```

Comment

The `CompareColumns` property is used with the TableCompare Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

<i>cell</i> , page 35 (<code>wlTable</code> and <code>row</code> property)	<i>cellIndex</i> , page 37 (<code>cell</code> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<code>wlTable</code> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<code>wlTable</code> property)	<i>InnerHTML</i> , page 153 (<code>cell</code> property)
<i>InnerText</i> , page 155 (<code>cell</code> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<code>wlTable</code> property)	<i>rowIndex</i> , page 225 (<code>row</code> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<code>cell</code> property)
<i>wlTable</i> , page 364	
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

CompareRows (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

TableCompare

Description

Specifies whether table comparison should be completed for all rows in the tables ["all"] or whether only the cells in a specific list of rows should be compared (for example, 1, 3, 6, 7, & 8, etc.).

Syntax

String ["all"] (default)

or

Array of any combination of:

- Integer row numbers [x, y, z, etc.]
- String range of row numbers ["1-3", "6-8", etc.]

Example

```
a.CompareRows = [1, 3, "6-8"]
```

Comment

The `CompareRows` property is used with the `TableCompare` Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

cell, page 35 (`wlTable` and `row` property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

id, page 143 (`wlTable` property)

InnerText, page 155 (`cell` property)

Prepare(), page 205

row, page 223 (`wlTable` property)

TableCompare, page 286

wlTable, page 364

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

cellIndex, page 37 (`cell` property)

cols, page 48 (`wlTable` property)

CompareColumns, page 51

Details, page 72

InnerHTML, page 153 (`cell` property)

MatchBy, page 178

ReportUnexpectedRows, page 216

rowIndex, page 225 (`row` property)

tagName, page 289 (`cell` property)

connectionBandwidth (property)

Property of Object

wlMediaPlayer

Description

Retrieves the connection bandwidth for the client in bits per second. (read-only double word)

Syntax

```
MyMediaPlayerObject.connectionBandwidth
```

Example

```
currentBandwidth = MyMediaPlayerObject.connectionBandwidth
```

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

ConnectionSpeed (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

Description

WebLOAD allows users to simulate various system and connection configurations, including setting a ‘virtual limit’ on the connection speed available during a test session. Obviously, the speed of the connection to a Web site is an important factor in the response time seen by users. Setting a limit on the connection speed during a test session allows testers working with higher-speed connections within their own labs to test systems for clients that may be limited in their own workplace connection speeds.

By default, WebLOAD will work with the fastest available connection speed. Testers may set the connection speed to any slower value, measured in bits per second (bps). For example, users may set values of 14,400bps, 28,800bps, etc. Note that the typical single ISDN line can carry 64Kb, a double line carries 128Kb, and a T1 line can handle 1.5 meg.

Syntax

You may assign a connection speed using the `wlGlobals.ConnectionSpeed` property. For example:

```

InitAgenda ()
{
    wlGlobals.ConnectionSpeed=28800
}
// main Agenda body
wlHttp.Get ("http://abcdef")
Sleep (1000)

```

GUI mode

WebLOAD recommends setting the connection speed through the Console GUI. You may set different connection speed limits for both the Load Generator and the Probing Client through the checkboxes on the Connection tab of the Tools | Default Options dialog box, illustrated in the following figure:

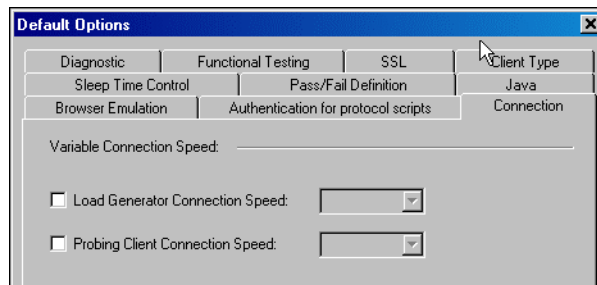


Figure 2-7: Setting Connection Speed Limits

See also

Browser configuration components, page 30

wlGlobals, page 339

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ContainerTag (property)

Property of Object

Div

Span

UIContainer

Description

Indicates type of UI container.

Syntax

<NA>

See also*Div*, page 77*Span*, page 254*UIContainer*, page 301

ContainingMap (property)

Property of Objects*area**map***Description**

Each `ContainingMap` object stores a pointer to the map in which an area object is found. The map objects are accessed through collections of `document.all`.

Syntax

<NA>

See also*area*, page 24*map*, page 178

content (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*wlMeta***Description**

Retrieves the value of the CONTENT attribute of the META tag (read-only string).

Syntax`wlMetas[index#].content`**Example**`document.wlMetas[0].content`

Comment

WebLOAD recommends accessing meta objects on a Web page through the standard `document.all` collection.

See also

content, page 56

httpEquiv, page 143

Name, page 184

Url, page 302

wlMeta, page 346

Coords (property)

Property of Object

area

Description

The coordinates of the area object.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

area, page 24

CopyFile() (function)

Description

Copies files from a source file to a destination file. The destination file is either explicitly or automatically named. `CopyFile` can copy both text and binary data files.

Syntax

```
CopyFile(SrcFileName [, DestFileName])
```

Parameters

`SrcFileName`—A literal string or variable containing the full literal name of the file to be copied. Specific system directories are searched for the specified file in the search order described in the preceding section.

`DestFileName`—An optional literal string or variable containing the full literal name of the file into which the source file will be copied. If the target parameter is omitted, WebLOAD will copy the source file to the current directory and return the file name as the return value of the `CopyFile` function.

Return Value

Optionally, a string with the target file name, returned if the `DestFileName` parameter is not specified.

Example

To copy the auxiliary file `src.txt`, located on the Console, to the destination file `dest.txt` on the current Load Generator, use the following command:

```
function InitAgenda() {
    ...
    CopyFile("src.txt", "dest.txt")
    ...
}
```

You may then access the file as usual in the main body of the Agenda. For example:

```
DataArr = GetLine("dest.txt")
```

It is convenient to specify only the `SrcFileName`. To copy the auxiliary file `file.dat`, located on the Console, to the current Load Generator, using a single file name:

```
function InitAgenda() {
    ...
    filename = CopyFile("file.dat")
    ...
}
...
GetLine(filename)
...
```

GUI mode

Note that `CopyFile()` and `IncludeFile()` functions can be added directly to the code in an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Select a function from the Edit | Insert | JavaScript | Copy/Include Files menu. The Visual AAT automatically inserts the correct code for the selected function into the Agenda file. The user may then edit parameter values without any concerns about mistakes in the function syntax.

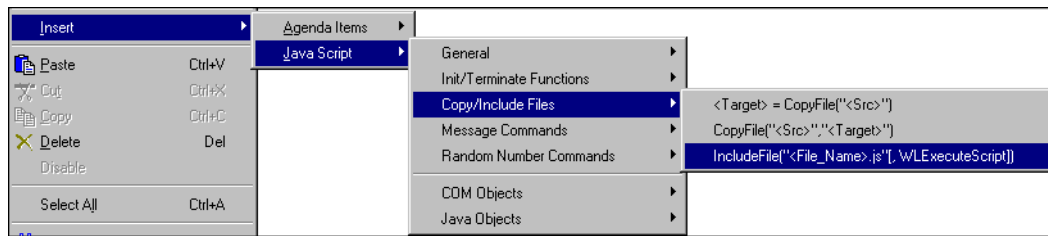


Figure 2-8: Inserting a CopyFile function

Comment

WebLOAD does not create new directories, so any directories specified as target directories *must already exist*.

The CopyFile command must be inserted in the InitAgenda () section of your JavaScript program.

See also

Close(), page 45

delete(), page 70

GetLine(), page 124

Open(), page 193

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

wlOutputFile(), page 351

Write(), page 372

CopyFile(), page 57

File management functions, page 100

IncludeFile(), page 150

Reset(), page 219

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile(), page 354

WriteIn(), page 373

CreateDOM() (function)

Mode

This function is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

CreateDOM functions return a complete Document Object Model (DOM) tree. A description of the WebLOAD DOM hierarchy is found in *Understanding the WebLOAD DOM structure*, page 215 in the *WebLOAD Programming Guide*. You may compare this expected DOM to the actual DOM generated automatically as your JavaScript Agenda runs.

Syntax

```
DOM = CreateDOM(HTMLFileName)
```

Parameters

HTMLFileName—A literal string or variable containing the full literal name of the HTML file in which the information about the expected DOM is found.

Return Value

Returns a complete Document Object Model (DOM) tree.

Example

```
DOM = CreateDOM("HTMLsource")
```

Comment

One of the most common practices in functional testing is to compare a known set of correct results previously generated by an application (expected data) to the results produced by an actual current execution of the application (actual data). These sets of results are stored in various Document Object Models (DOMs).

The actual DOM is created automatically each time an HTTP request is accessed through the document object. The expected DOM is assigned by the user to a specific HTTP command. To make the verification functions more easily readable, WebLOAD uses the alias ACTUAL to access the actual document and the alias EXPECTED to access the expected document.

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

CreateTable() (function)

Mode

This function is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

WebLOAD provides a `CreateTable` function to automatically convert the tables found on an HTML page to parallel `wlTable` objects. This simplifies access to the exact table entry in which the user is interested. The `CreateTable()` function returns a window object that includes a `wlTables` collection. This is a collection of `wlTable` objects, each of which corresponds to one of the tables found on the HTML page used as the function parameter. The table data may be accessed as any standard `wlTable` data.

Syntax

```
CreateTable(HTMLFileName)
```

Parameters

`HTMLFileName`—A literal string or variable containing the full literal name of the HTML file in which the tables to be converted are found.

Return Value

Returns a window object that includes a `wlTables` collection.

Example

```
NewTableSet = CreateTable("HTMLTablePage")
NumTables = NewTableSet.wlTables.length
FirstTableName = NewTableSet.wlTables[0].id
```

Comment

`CreateTable()` is a member of the `wlTables` family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide* *BeginTransaction()*, page 28

- CreateDOM()*, page 59
- Custom verification functions*, page 137 in the *WebLOAD Programming Guide*
- EndTransaction()*, page 85
- ReportEvent()*, page 214
- TableCompare*, page 286
- Transaction verification components*, page 297
- Validate()*, page 309
- Verification Test Property List: Global and Page Level*, page 318
- wlBrowser*, page 325
- Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*
- CreateTable()*, page 60
- Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*
- Functional Testing and Reporting*, page 127 in the *WebLOAD Programming Guide*
- SetFailureReason()*, page 239
- TimeoutSeverity*, page 293
- TransactionTime*, page 298
- Verification Test Components*, page 315
- VerificationFunction()* (user-defined), page 320
- wlTable*, page 364

currentPosition (property)

Property of Object

wlMediaPlayer

Description

Retrieves the current position in the stream in milliseconds from the beginning. (read-only double number)

Syntax

`MyMediaPlayerObject.currentPosition`

Example

<NA>

See also

- bitrate*, page 30
- connectionBandwidth*, page 53
- currentPosition*, page 62
- currentStreamName*, page 62
- duration*, page 80
- fileName*, page 100
- OpenStream()*, page 194
- Pause()*, page 200
- Play()*, page 200
- Resume()*, page 220
- state*, page 277
- Stop()*, page 278

type, page 300

wlMediaPlayer, page 344

currentStreamName (property)

Property of Object

wlMediaPlayer

Description

Specifies the name of the currently playing stream. (read string)

Syntax

`MyMediaPlayerObject.currentStreamName`

Example

<NA>

Comment

When using a redirection file, the value of `currentStreamName` will not be the same as the value of `fileName`.

See also

bitrate, page 30

connectionBandwidth, page 53

currentPosition, page 62

currentStreamName, page 62

duration, page 80

fileName, page 100

OpenStream(), page 194

Pause(), page 200

Play(), page 200

Resume(), page 220

state, page 277

Stop(), page 278

type, page 300

wlMediaPlayer, page 344

Data (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

Holds a string to be submitted in an HTTP Post command. The `Data` property has two subfields:

`Data.Type`—the MIME type for the submission

`Data.Value`—the string to submit

You can use `Data` in two ways:

- ◆ As an alternative to `FormData` if you know the syntax of the form submission.
- ◆ To submit a string that is not a standard HTML form and cannot be represented by `FormData`.

Example

Thus the following three code samples are equivalent:

```
//Sample 1
wlHttp.Data.Type = "application/x-www-form-urlencoded"
wlHttp.Data.Value = "SearchFor=icebergs&SearchType=ExactTerm"
wlHttp.Post ("http://www.ABCDEF.com/query.exe")

//Sample 2
wlHttp.FormData.SearchFor = "icebergs"
wlHttp.FormData.SearchType = "ExactTerm"
wlHttp.Post ("http://www.ABCDEF.com/query.exe")

//Sample 3
wlHttp.Post ("http://www.ABCDEF.com/query.exe" +
             "?SearchFor=icebergs&SearchType=ExactTerm")
```

Methods

`wlClear()`, page 328

Properties

type, page 300

value, page 310

Comment

`Data` and `DataFile` are both collections that hold sets of data. `Data` collections are stored within the `Agenda` itself, and are useful when you prefer to see the data directly. `DataFile` collections store the data in local text files, and are useful when you are working with large amounts of data, which would be too cumbersome to store within the `Agenda` code itself. When working with `DataFile` collections, only the name of the text file is stored in the `Agenda` itself.

Your Agenda should work with either `Data` or `DataFile` collections. Do not use both properties within the same Agenda.

See also

Data, page 63

Erase, page 87

FormData, page 104

Header, page 138

wlHttp, page 342

DataFile, page 65

fileName, page 100

Get(), page 110

Post(), page 202

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

DataFile (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

A file to be submitted in an HTTP Post command.

WebLOAD sends the file using a MIME protocol. `DataFile` has two subfields:

- ◆ `DataFile.Type`—the MIME type
- ◆ `DataFile.FileName`—the name of the file, for example, `"c:\\MyWebloadData\\BigFile.doc"`

WebLOAD sends the file when you call the `wlHttp.Post()` method.

Methods

wlClear(), page 328

Properties

fileName, page 100

type, page 300

Comment

`Data` and `DataFile` are both collections that hold sets of data. `Data` collections are stored within the Agenda itself, and are useful when you prefer to see the data directly. `DataFile`

collections store the data in local text files, and are useful when you are working with large amounts of data, which would be too cumbersome to store within the Agenda code itself. When working with `DataFile` collections, only the name of the text file is stored in the Agenda itself.

Your Agenda should work with either `Data` or `DataFile` collections. Do not use both properties within the same Agenda.

See also

Data, page 63

Erase, page 87

Get(), page 110

Post(), page 202

wlHttp, page 342

DataFile, page 65

FormData, page 104

Header, page 138

value, page 310

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

DefaultAuthentication (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlLocals

wlHttp

Description

This property specifies which user authentication protocol WebLOAD should use to log onto an HTTP server. The possible values are:

- ◆ Basic—WebLOAD sends the `UserName` and `Password`. (default)
- ◆ NT Challenge/Response—WebLOAD sends the `NTUserName` and `NTPassword`. If these are empty, WebLOAD sends the Windows NT user name and password under which the Agenda is running. WebLOAD always sends the appropriate responses to the Server authentication challenges.

Syntax

<NA>

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box, illustrated below.

These two GUI approaches are both illustrated in the *WebLOAD Programming Guide*.

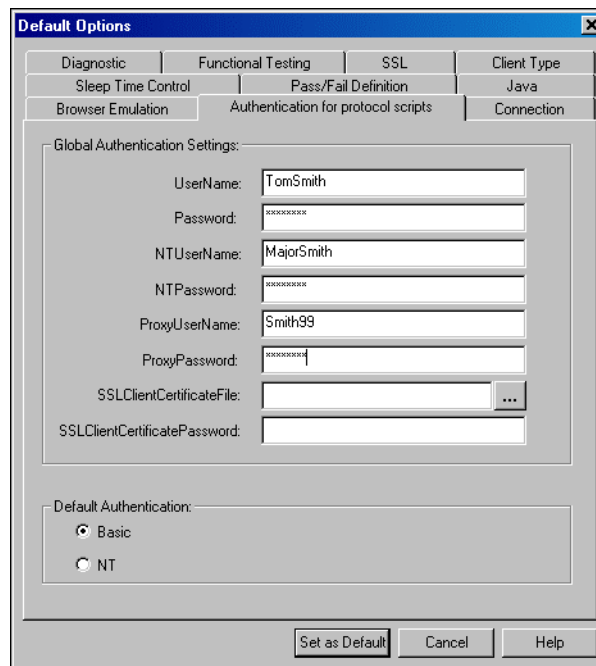


Figure 2-9: Setting Authentication values

See also

Browser configuration components, page 30

wlGlobals, page 339

wlLocals, page 343

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

defaultchecked (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

element

Description

For an `<INPUT type="checkbox">` or `<INPUT type="radio">` element, the default checked value of the form element (read-only string).

Syntax

`<NA>`

See also

AdjacentText, page 23

checked, page 39

defaultchecked, page 68

element, page 83

InnerText, page 155

Name, page 184

row, page 223

Size, page 247

type, page 300

value, page 310

Alt, page 23

cols, page 48

defaultvalue, page 69

id, page 143

MaxLength, page 180

option, page 195

selectedindex, page 234

title, page 296

Url, page 302

defaultselected (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

option

Description

Returns a Boolean value specifying whether this option was the one originally "selected" before any user acted upon this "select" control.

Syntax

<NA>

See also

defaultchecked, page 68

option, page 195

selected, page 233

text, page 291

value, page 310

defaultvalue (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

element

Description

The default value of the form element (read-only string).

Syntax

<NA>

See also

element, page 83

delete() (method)

Method of Objects

wlOutputFile

wlCookie

Description

This method deletes the items to which the parent object points.

If the parent is a built-in `wlCookie` object, delete all cookies set by `wlCookie` in the current thread.

If the parent is a user-created `wlOutputFile` object, deletes the `wlOutputFile` object and closes the output file.

Syntax

wlCookie:

When working with `wlCookie` objects, use the uppercase form:

```
wlCookie.Delete(name, domain, path)
```

wlOutputFile:

When working with `wlOutputFile` objects, use the lowercase form:

```
wlOutputFile.delete()
```

Parameters

wlCookie:

`name`—A descriptive name used for the cookie to be deleted, for example, "CUSTOMER".

`domain`—The top-level domain name for the cookie being deleted, for example, "www.ABCDEF.com".

`path`—The top-level directory path, within the specified domain, for the cookie being deleted, for example, "/".

wlOutputFile:

None.

Return Value

None.

Example

wlCookie:

```
wlCookie.Delete("CUSTOMER", "www.ABCDEF.com", "/")
```

wlOutputFile:

```
MyFileObj = new wlOutputFile(filename)
```

```
...
```

```
MyFileObj.delete()
```

Comment

Note that the `wlCookie` property is written in uppercase: `wlCookie.Delete()`.

See also

<i>Close()</i> , page 45	<i>CopyFile()</i> , page 57
<i>delete()</i> , page 70	<i>File management functions</i> , page 100
<i>GetLine()</i> , page 124	<i>IncludeFile()</i> , page 150
<i>Open()</i> , page 193	<i>Reset()</i> , page 219
<i>Using the Form Data Wizard</i> , page 35 in the <i>WebLOAD Programming Guide</i>	<i>Using the IntelliSense JavaScript Editor</i> , page 18
<i>wlOutputFile()</i> , page 351	<i>wlOutputFile()</i> , page 354
<i>Write()</i> , page 372	<i>Writeln()</i> , page 373

See also

<i>Close()</i> , page 45	<i>CopyFile()</i> , page 57
<i>delete()</i> , page 70	<i>File management functions</i> , page 100
<i>GetLine()</i> , page 124	<i>IncludeFile()</i> , page 150
<i>Open()</i> , page 193	<i>Reset()</i> , page 219
<i>Using the Form Data Wizard</i> , page 35 in the <i>WebLOAD Programming Guide</i>	<i>wlCookie</i> , page 330
<i>wlOutputFile()</i> , page 351	<i>wlOutputFile()</i> , page 354
<i>Write()</i> , page 372	<i>Writeln()</i> , page 373

Details (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

TableCompare

Description

Specifies whether table comparison should stop at the first error found (`false`) or whether the function should complete a full comparison of all selected table cells and save information about the details of all errors found (`true`).

Syntax

Boolean (`true/false`, default `false`)

Example

```
a.Details = true
```

Comment

The `Details` property is used with the TableCompare Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

<i>cell</i> , page 35 (<code>wlTable</code> and <code>row</code> property)	<i>cellIndex</i> , page 37 (<code>cell</code> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<code>wlTable</code> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<code>wlTable</code> property)	<i>InnerHTML</i> , page 153 (<code>cell</code> property)
<i>InnerText</i> , page 155 (<code>cell</code> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<code>wlTable</code> property)	<i>rowIndex</i> , page 225 (<code>row</code> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<code>cell</code> property)
<i>wlTable</i> , page 364	
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

Dialog box properties

Properties of Objects

wlBrowser

Description

Web sessions often include some sort of dialog box interaction, either expected (such as login or confirmation) or unexpected (such as error or warning). WebLOAD Agendas automatically handle both expected and unexpected dialog boxes appropriately during the course of a test session, through a set of dialog box properties attached to the `wlBrowser` object.

Each type of dialog box type is actually a separate object property, each of which includes its own distinct sub-property list, depending on the nature of the dialog box. Users who are editing the JavaScript code in an Agenda may set dialog box properties manually. Table 1, page 74, lists all the dialog box types, their properties, and an example of JavaScript syntax for each box.

Syntax

```
wlBrowser.DialogBoxType.SubProperty = "PropertyValue"
```

Example

Many Web products include popup confirmation boxes in which the user clicks on either a CONFIRM or a CANCEL button. These are referred to as dialog boxes of type `Confirm`, with the single sub-property `Answer`, which may have a value of either `OK` or `Cancel`. In a JavaScript Agenda, the default value for such a dialog box would be recorded as either:

```
wlBrowser.Confirm.Answer = "OK"
```

or

```
wlBrowser.Confirm.Answer = "Cancel"
```

GUI mode

In most cases WebLOAD handles dialog boxes, both expected and unexpected, automatically, based on built-in default values. WebLOAD recommends customizing these default settings through the Visual AAT or Console GUI. Select values from the drop down lists in the Browser Dialogs tab of the Tools | Default or Current Project Options dialog box, illustrated below:

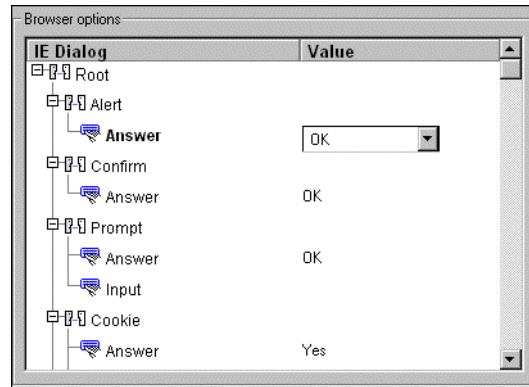


Figure 2-10: Managing Dialog Boxes

Properties

The following table lists all dialog box types currently available through WebLOAD, together with their properties and an example of JavaScript syntax for each box.

Table 2-1: Dialog Box Property List

Dialog Box Type	Property and Value	Syntax Examples
Alert	Answer=OK	<code>wlBrowser.Alert.Answer="OK"</code>

Note: During WebLOAD testing, an Alert dialog box does not literally trigger an alert. It simply sends an InfoMessage to the Log window.

Dialog Box Type	Property and Value	Syntax Examples
Client Authentication	Username=<string> Password=<string> Domain=<string> optional-indicates NT Chalange/Response Answer=OK/Cancel Save=<string>	wlBrowser.ClientAuthentication.Username ="string" wlBrowser.ClientAuthentication.Password ="string" wlBrowser.ClientAuthentication.Domain ="string" wlBrowser.ClientAuthentication.Answer ="OK" wlBrowser.ClientAuthentication.Save ="unchecked"
Client Certificate	Answer=OK/Cancel Certificate=<string>	wlBrowser.ClientCertificate.Answer ="Cancel" wlBrowser.ClientCertificate.Certificate ="0"
Confirm	Answer=OK/Cancel	wlBrowser.Confirm.Answer="OK"
Cookie	Answer=Yes/No NeverShow=<string>	wlBrowser.Cookie.Answer="No" wlBrowser.Cookie.NeverShow="unchecked"
Ftp Authentication	Answer=Login/Cancel Server=<string> Username=<string> Password=<string> Email=<string> Anonymous=<string> If checked, no need for Username/Password Save=<string>	wlBrowser.FtpAuthentication.Answer= "Login" wlBrowser.FtpAuthentication.Server= "string" wlBrowser.FtpAuthentication.Username= "string" wlBrowser.FtpAuthentication.Password= "string" wlBrowser.FtpAuthentication.Email= "string" wlBrowser.FtpAuthentication.Anonymous ="unchecked" wlBrowser.FtpAuthentication.Save ="unchecked"
Ok	Answer=OK/Cancel	wlBrowser.Ok.Answer="OK"
OkCancel	Answer=OK/Cancel	wlBrowser.OkCancel.Answer="OK"
Prompt	Input=<string> Answer=OK/Cancel	wlBrowser.Prompt.Input="string" wlBrowser.Prompt.Answer="OK"
<p>Note: During WebLOAD testing, the input from a Prompt dialog box is used to set a property value. Output is simply sent to the Visual AAT or Console Log window, rather than triggering a new output popup window.</p>		
YesNo	Answer=Yes/No	wlBrowser.YesNo.Answer="Yes"
YesNoCancel	Answer=Yes/No	wlBrowser.YesNoCancel.Answer="Yes"

See also

Dialog Boxes, page 75 in the *WebLOAD Programming Guide*

wlBrowser, page 325

DisableSleep (property)

Property of Objects

wlBrowser

Description

Boolean flag that indicates whether the recorded sleep pauses will be included in the test session.

Syntax

```
wlBrowser.DisableSleep = True           //Ignore recorded sleep periods
wlBrowser.DisableSleep = False         //Include recorded sleep periods
```

Comment

Sleep periods during test sessions are by default kept to the length of the sleep period recorded by the user during the original recording session. If you wish to include sleep intervals but change the time period, set `DisableSleep` to `False` and assign values to the other sleep properties as follows:

- ◆ `SleepRandomMin`—Assign random sleep interval lengths, with the minimum time period equal to this property value.
- ◆ `SleepRandomMax`—Assign random sleep interval lengths, with the maximum time period equal to this property value.
- ◆ `SleepDeviation`—Assign random sleep interval lengths, with the time period ranging between this percentage value more or less than the original recorded time period.

GUI mode

WebLOAD recommends setting the sleep mode through the Visual AAT or Console GUI. Choose a setting from the Sleep Time Control tab of the Tools | Default, Current or Agenda Options dialog box, illustrated below:

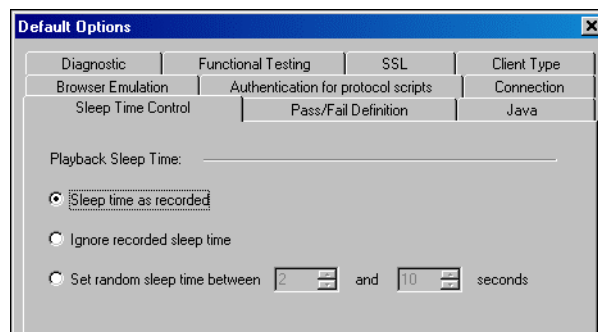


Figure 2-11: Setting the sleep mode

See also

Sleeping or pausing in mid-session, page 28 in the *WebLOAD Programming Guide*

DisableSleep, page 75*Sleep()*, page 247*SleepDeviation*, page 250*SleepRandomMax*, page 251*SleepRandomMin*, page 252*wlBrowser*, page 325

Div (object)

Property of Objects

Specifies a container that renders HTML. `Div` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Each `Div` object represents one of the `<DIV>` user interface fields embedded in a document. One of the set of `UIContainer` objects. (Compare to the `element` object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the `form` object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["DIV"]
```

Methods

wlClick(), page 326*wlMouseDown()*, page 347*wlMouseOver()*, page 348*wlMouseUp()*, page 349*wlMultiSelect()*, page 350*wlSelect()*, page 358*wlTypeIn()*, page 366

Properties

ContainerTag, page 55*event*, page 91*id*, page 143*InnerText*, page 155*OnClick*, page 190*OnMouseOver*, page 191

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99

<i>form</i> , page 102	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>length</i> , page 167	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301

DNSUseCache (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlHttp

wlLocals

Description

Enable caching of IP addresses that WebLOAD receives from a domain name server. The value of `DNSUseCache` may be:

- ◆ No—Disable IP address caching.
- ◆ Yes—Enable IP address caching. (default)

Assign a "Yes" value to reduce the time for domain name resolution. Assign a "No" value if you want to include the time for name resolution in the WebLOAD performance statistics.

Syntax

<NA>

GUI mode

WebLOAD recommends enabling or disabling the DNS cache through the Console GUI. Enable caching for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

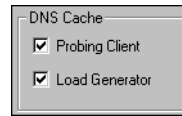


Figure 2-12: Enabling DNS Cache for Load Generator

Comment

To clear the DNS cache when working in HTTP Protocol mode, set the *ClearDNSCache()* property, described on page 41.

See also

Browser configuration components, page 30

ClearSSLCache(), page 42

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlLocals, page 343

ClearDNSCache(), page 41

DNSUseCache, page 78

SSLUseCache, page 273

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

document (object)

Property of Object

The `document` object is a property of the following object:

window

Description

Represents the HTML document in a given browser window. The `document` object is one of the main entry points into the DOM, used to retrieve parsed HTML data. `document` objects store the complete parse results for downloaded HTML pages. Use the `document` properties to retrieve links, forms, nested frames, and other information about the document.

`document` objects are local to a single thread. WebLOAD creates an independent `document` object for each thread of an Agenda. You cannot create new `document` objects using the JavaScript `new` operator, but you can access HTML documents through the properties and methods of the standard DOM objects. `document` properties are read-only.

Syntax

Access all elements of the Browser DOM through the `document` object, using the standard syntax. For example:

```
//Store the text of an entire page into a variable, myMainPage
myMainPage = document.documentElement.innerText
```

WebLOAD Agendas work with all browser DOM collections and objects. The recommended way to access these objects is through the classic browser `document` object, via the `document.all` property, by checking the tag type. For example, access a table through:

```
document.all.tags["TABLE"]
```

Example

If you are working in the HTTP Protocol mode, then to access the form's element fields use the following expression:

```
document.frames[1].forms[0].elements[#].<property>
```

For example:

```
document.frames[1].forms[0].elements[1].type
```

Methods

wlGetAllForms(), page 336

wlGetAllFrames(), page 337

wlGetAllLinks(), page 338

Properties

form, page 102

frames, page 107

Image, page 148

link, page 169

location, page 176

script, page 229

title, page 296

wlHeader, page 340

wlMeta, page 346

wlSource, page 360

wlStatusLine, page 361

wlStatusNumber, page 362

wlTable, page 364

wlVersion, page 368

wlXmIs, page 370

See also

window, page 324

duration (property)

Property of Object

wlMediaPlayer

Description

Retrieves the duration of the current stream in milliseconds.
(read-only long number)

Syntax

```
MyMediaPlayerObject.duration
```

Example

```
<NA>
```

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

Dynamic Object Recognition (DOR) components

Properties and Methods of Object

wlBrowser

wlHttp

Description

WebLOAD provides a high-level working environment for users, recording any objects of interest to users without the user needing to be aware of the objects' underlying DOM properties or identification details. During subsequent playback test sessions, some of the recorded objects' properties may change slightly, due to the dynamic nature of Web activity. For example, an object's location or URL may be different from that originally recorded. WebLOAD provides full access to the complete set of objects originally of interest to the users at recording time, even if they include a slightly altered set of DOM properties. Through DOR, users are able to specify which DOM properties of a selected object are of greater interest or significance and which properties to ignore when searching for an object match.

Syntax

```
wlBrowser.ObjectProperty["FieldName"] = "FieldValue"
MyObject = wlBrowser.FindObject
    (<object type>, <object index> [, "<object name>"])
```

Example

The general syntax for a typical DOR code block is:

```
// Reset focus to new window with wlHttp.SetWindow() or wlHttp.Navigate()
wlBrowser.ObjectProperty["<property name1>"] = <property value1>
wlBrowser.ObjectProperty["<property name2>"] = <property value2>
obj = wlBrowser.FindObject(<object type>, <object index> [, "<object name>"])
obj.method()
```

For example:

```
wlBrowser.Navigate("http://www.abc.com")
wlBrowser.ObjectProperty["Location"] = "bottom#1.left#0"
wlBrowser.ObjectProperty["Url"] = "http://www.abc.com/gifts.htm"
wlBrowser.ObjectProperty["Text"] = "GIFTS"
link1 = wlBrowser.FindObject(FT_LINK, 2)
link1.wlClick()
```

Sometimes an object is identified using another object. For example, you might use an image to identify its parent link. In this case the identifying properties of the other object are added to the list. For example:

```
wlBrowser.Navigate("http://www.abc.com")
wlBrowser.ObjectProperty["Image_Url"] = "http://www.abc.com/images/prod.gif"
wlBrowser.ObjectProperty["Location"] = "bottom#1.left#0"
wlBrowser.ObjectProperty["Url"]="http://www.abc.com/products.htm"
link1 = wlBrowser.FindObject(FT_LINK, 5)
link1.wlClick()
```

Properties and Methods

FindObject(), page 102

ObjectProperty[], page 189

Comment

Agendas that work within the HTTP Protocol mode use a slightly different approach to Automatic State Management. See *Automatic State Management for HTTP Protocol Mode*, page 25, in this manual, and *Automatic State Management (ASM) in HTTP Mode*, page 223 in the *WebLOAD Programming Guide* for more information.

See also

Automatic State Management for HTTP Protocol Mode, page 25

Automatic State Management (ASM) in HTTP Mode, page 223 in the *WebLOAD Programming Guide*

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

Dynamic Object Recognition (DOR) components, page 81

FindObject(), page 102

Navigate(), page 186

ObjectProperty[], page 189

SetWindow(), page 244

wlBrowser, page 325*wlHttp*, page 342

element (object)

Property of Object

`element` objects are grouped into collections of `elements`. The `elements` collection is also a property of the following objects:

form

Description

Each `element` object stores the parsed data for a single HTML form element such as `<INPUT>`, `<BUTTON>`, `<TEXTAREA>`, or `<SELECT>`. The full `elements` collection stores all the controls found in a given form except for objects of `input type=image`. (Compare to the `form` object, which stores the parsed data for an entire HTML form.)

`element` objects are local to a single thread. You cannot create new `element` objects using the JavaScript `new` operator, but you can access HTML elements through the properties and methods of the standard DOM objects. `element` properties are read-only.

Syntax

`element` objects are organized into collections of `elements`. `elements[0]` refers to the first child element, `elements[1]` refers to the second child element, etc. To access an individual element's properties, check the `length` property of the `elements` collection and use an index number to access the individual elements. For example, to find out how many `element` objects are contained within `forms[1]`, check the value of:

```
document.forms[1].elements.length
```

You can access a member of the `elements` collection either by its index number or by its HTML name attribute. For example, suppose that the first element of a form is coded by the HTML tag

```
<INPUT type="text" name="yourname">
```

You can access this element by writing either of the following expressions:

```
document.forms[0].elements[0]
document.forms[0].elements["yourname"]
document.forms[0].elements.yourname
document.forms[0].yourname
```

Example

Access each element's properties directly using either of the following lines:

```
document.forms[0].elements[0].type
```

OR

`document.forms[0].yourname.type`

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlTypeIn(), page 366

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

AdjacentText, page 23

checked, page 39

defaultchecked, page 68

InnerText, page 155

Coords, page 57

InnerImage, page 154

MaxLength, page 180

OnClick, page 190

option, page 195

Size, page 247

id, page 143

option, page 195

selectedIndex, page 234

title, page 296

Url, page 302

Alt, page 23

cols, page 48

defaultvalue, page 69

Name, page 184

id, page 143

InnerText, page 155

Name, page 184

OnMouseOver, page 191

OuterLink, page 196

Shape, page 246

MaxLength, page 180

row, page 223

Size, page 247

type, page 300

value, page 310

Comment

The most frequently accessed input elements are of type Button, CheckBox, File, Image, Password, Radio, Reset, Select, Submit, Text, and TextArea.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputRadio, page 162

InputText, page 164

length, page 167

Radiobutton, page 210

Select, page 231

text, page 291

TextArea, page 292

encoding (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

form

Description

A read-only string that specifies the MIME encoding for the form.

Syntax

<NA>

See also

form, page 102

EndTransaction() (function)

Mode

This function is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

Use the `BeginTransaction()` and `EndTransaction()` functions to define the start and finish of a logical block of code that you wish to redefine as a single logical transaction unit. This enables setting timers, verification tests, and other measurements for this single logical unit.

Syntax

```

BeginTransaction(TransName)
...
  <any valid JavaScript code>
...
EndTransaction(TransName, Verification, [SaveFlag])

```

Parameters

TransName—The name assigned to this transaction, a user-supplied string.

Verification—A call to any verification function that returns one of the following values: `WLSuccess`, `WLMinorError`, `WLError`, or `WLSevereError`. If the verification function does not explicitly return a value, the default value is always `WLSuccess`. Verification may also be an expression, constant, or variable that evaluates to one of the preceding return values. See *VerificationFunction() (user-defined)*, page 320, for more information.

[SaveFlag]—An optional Boolean flag specifying whether WebLOAD should save the results of *all transaction instances*, successes and failures, (`true`), for later analysis with Data Drilling, or should save only results of *failed transaction instances* that triggered some sort of error flag, (`false`, default). (See *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*, for more information about Data Drilling.)

Return Value

None

Example

<NA>

GUI mode

Note that `BeginTransaction()` and `EndTransaction()` functions are usually accessed and inserted into Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates a Form branch in the Agenda Tree bracketed by `BeginTransaction` and `EndTransaction` nodes. The `EndTransaction` node is highlighted in the Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right. See *Adding transactions*, page 49 in the *WebLOAD Programming Guide*, for more information.

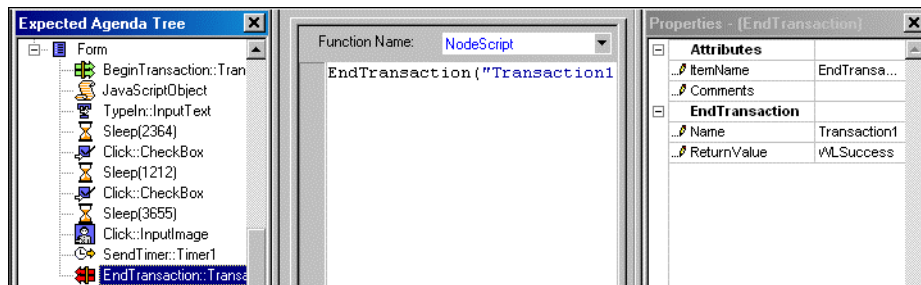


Figure 2-13: EndTransaction added to an Agenda

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

Erase (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

Indicates whether or not to clear the WebLOAD properties of a `wlHttp` object after each `Get()`, `Post()`, or `Head()` call. `wlHttp.Erase` is a read/write property. The default value is `true`. This section briefly discusses the implications of each setting.

`wlHttp.Erase=true` (default)

When `Erase` is set to `true`, WebLOAD automatically erases all `wlHttp` property values after each HTTP access. You must reassign any properties you need before the next HTTP access. For this reason, assign the properties of `wlHttp` only in the *main script*, not in `InitClient()`, so they will be reassigned in every round.

Thus if `Erase` is set to `true` the following Agenda is incorrect. In this Agenda, the `wlHttp` properties are assigned values in `InitClient()`. The Agenda would connect to the `Url` and submit the `FormData` only in the first round. After the first `Post()` call, the `Url` and `FormData` property values are erased, so WebLOAD cannot use them in subsequent rounds.

```
function InitClient() { //Wrong!
    wlHttp.Url = "http://www.ABCDEF.com/products.exe"
    wlHttp.FormData["Name"] = "John Smith"
    wlHttp.FormData["Product Interest"] = "Modems"
}
//Main script
wlHttp.Post()
```

To solve the problem, assign the `wlHttp` property values in the **main script**, so that the assignments are executed before each `Get()`, `Post()`, or `Head()` call:

```
//Main script //OK
wlHttp.Url = "http://www.ABCDEF.com/products.exe"
wlHttp.FormData["Name"] = "John Smith"
wlHttp.FormData["Product Interest"] = "Modems"
wlHttp.Post()
```

Alternatively, you could assign values to **wlLocals properties**, which are not erased:

```
function InitClient() { //OK
    wlLocals.Url = "http://www.ABCDEF.com/products.exe"
    wlLocals.FormData["Name"] = "John Smith"
    wlLocals.FormData["Product Interest"] = "Modems"
}
//Main script
wlHttp.Post()
```

`wlHttp.Erase=false`

You may set `Erase` to `false` to prevent erasure. For example, if for some reason you absolutely had to assign values to the `wlHttp` properties in the `InitClient()` function of the Agenda, change the value of the `Erase` property to `false`. If `Erase` is `false`, the properties retain their values through subsequent rounds.

Thus another way to correct the preceding example is to write:

```
function InitClient() { //OK
  wlHttp.Erase = false
  wlHttp.Url =
    "http://www.ABCDEF.com/products.exe"
  wlHttp.FormData["Name"] = "John Smith"
  wlHttp.FormData["Product Interest"] = "Modems"
}
//Main script
wlHttp.Post()
```

User-defined properties are not linked to the `wlHttp.Erase` property and will not be erased automatically by WebLOAD. The only way to reset or erase user-defined properties is for the user to set the new values explicitly.

See also

Data, page 63

DataFile, page 65

DataCollection.type, page 300

DataCollection.value, page 310

Erase, page 87

fileName, page 100

FormData, page 104

Get(), page 111

Header, page 138

Post(), page 202

wlClear(), page 328

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 of the *WebLOAD Programming Guide*

ErrorMessage() (function)

Description

Use this function to display an error message in the Log Window and abort the current round.

Syntax

```
ErrorMessage(msg)
```

Parameters

`msg`—A string with an error message to be sent to the Console.

Return Value

None.

Example

<NA>

Comment

If you call `ErrorMessage()` in the main script, WebLOAD stops the current round of execution and runs the standard error handling functions (`TerminateClient()`, etc.), if they exist in the Agenda. Execution continues with the next round, at the beginning of the main script.

You may also use the `wlException` object with the built-in `try()/catch()` commands to catch errors within your Agenda. For more information about error management options in WebLOAD, see *Error Management*, page 104 in the *WebLOAD Programming Guide*.

GUI mode

WebLOAD recommends adding message functions to your Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a Message Node to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

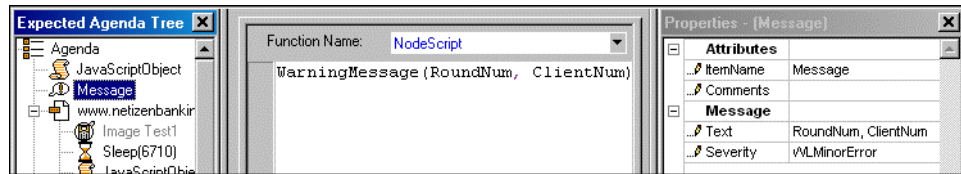


Figure 2-14: Adding a Message Node to an Agenda

See also

- | | |
|--|---|
| <i>Error Management</i> , page 104 in the <i>WebLOAD Programming Guide</i> | <i>ErrorMessage()</i> , page 89 |
| <i>ExceptionEnabled</i> , page 92 | <i>GetMessage()</i> , page 129 |
| <i>GetSeverity()</i> , page 133 | <i>InfoMessage()</i> , page 152 |
| <i>LiveConnect Overview</i> , page 255 in the <i>WebLOAD Programming Guide</i> | <i>Message functions</i> , page 180 |
| <i>ReportLog()</i> , page 215 | <i>SevereErrorMessage()</i> , page 245 |
| <i>Using the IntelliSense JavaScript Editor</i> , page 18 | <i>WarningMessage()</i> , page 323 |
| <i>wlException</i> , page 331 | <i>wlException()</i> object constructor, page 333 |

EvaluateScript() (function)

Description

Special purpose function. Allows testers to include scripts from an external library and specify the point during Agenda execution at which the script should be executed.

Syntax

```
EvaluateScript("ScriptName", RunModeConstant)
```

Parameters

ScriptName—A string or variable containing the full literal name of the file to be included.

RunModeConstant—One of the following list of constants that acts as a flag when passed as a parameter to `EvaluateScript()`. Defines the point during Agenda execution at which WebLOAD should execute the script being included here. Possible choices include:

- `WLAFTERINITAGENDA`
- `WLBFOREINITCLIENT`
- `WLBFORETHREADACTIVATION`
- `WLONTHEADACTIVATION`
- `WLBFOREROUND`
- `WLAFTERROUND`
- `WLAFTERTERMINATECLIENT`
- `WLAFTERTERMINATEAGENDA`

Return Value

None.

Example

<NA>

Comment

Should be included in the `InitAgenda()` function.

See also

Agenda execution sequence, page 65 in the *WebLOAD Programming Guide*

event (property)

Property of Objects

*Button**link**Span**UIContainer**Div**script**TableCell*

Description

Represents the event that occurred to the parent object or the event for which the script is written. When working with WebLOAD, the most commonly recorded events are *OnClick*, page 190, and *OnMouseOver*, page 191. For example, if the user clicks with the left mouse button on a *Button*, the event is *OnClick*. If the user moves the mouse pointer into an *Image*, the event is *OnMouseOver*.

Example

<NA>

See also

Actions, page 20*Back()*, page 27*Forward()*, page 106*OnClick*, page 190*Refresh()*, page 213*wlBrowser*, page 325*wlClose()*, page 329*wlMouseOver()*, page 348*wlMultiSelect()*, page 350*wlSubmit()*, page 362*AutoNavigate()*, page 26*event*, page 91*Navigate()*, page 186*OnMouseOver*, page 191*SetWindow()*, page 244*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseUp()*, page 349*wlSelect()*, page 358*wlTypeIn()*, page 366

ExceptionEnabled (property)

Property of Object

wlBrowser

Description

Flag that enables using the built-in `wlException` object for error management in the Agenda.

Example

```
InitAgenda () {
    ...
    wlBrowser.ExceptionEnabled=True
}
```

Comment

`wlBrowser.ExceptionEnabled` must be set to `True` in the `InitAgenda()` function to be able to use the `wlException` object later in the Agenda. WebLOAD by default always sets this property to `True`.

See also

<i>Error Management</i> , page 104 in the <i>WebLOAD Programming Guide</i>	<i>ErrorMessage()</i> , page 89
<i>ExceptionEnabled</i> , page 92	<i>GetMessage()</i> , page 129
<i>GetSeverity()</i> , page 133	<i>InfoMessage()</i> , page 152
<i>LiveConnect Overview</i> , page 255 in the <i>WebLOAD Programming Guide</i>	<i>Message functions</i> , page 180
<i>ReportLog()</i> , page 215	<i>SevereErrorMessage()</i> , page 245
<i>Using the IntelliSense JavaScript Editor</i> , page 18	<i>WarningMessage()</i> , page 323
<i>wlException</i> , page 331	<i>wlException()</i> object constructor, page 333

ExpectedDocument (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

`wlHttp`

Description

A pointer to the expected DOM structure in the Expected DOM Repository file that corresponds to the actual DOM structure created by the most recent HTTP transaction.

Syntax

```
wlHttp.ExpectedDocument = DOMPointer
```

Comment

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectedDOMID (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

A key that identifies the expected DOM structure in the Expected DOM Repository file that corresponds to the actual DOM structure created by the most recent HTTP transaction.

Syntax

```
wlHttp.ExpectedDOMID = DOMKey
```

Comment

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectedID (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The HTML session ID value that the WebLOAD DOM has stored for the current object. Note that this is the *expected* session ID. WebLOAD uses the `IdentifyObject` method to verify whether or not this is the actual session ID value at run time.

Syntax

```
wlHttp.ExpectedID = "OptionalIDString"
```

Comment

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

IdentifyObject(), page 147

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectedLocation (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The parsed URL data that the WebLOAD DOM has stored for the current object. Note that this is the *expected* location. WebLOAD uses the `IdentifyObject` method to verify whether or not this is the actual location value at run time.

Syntax

```
wlHttp.ExpectedLocation = "TargetShorthand"
```

Example

```
wlHttp.ExpectedLocation = ":#1.#1:"
```

Comment

`ExpectedLocation` uses the WebLOAD shorthand notation described in *Identifying frame locations*, page 224 in the *WebLOAD Programming Guide*.

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

Identifying frame locations, page 224 in the *WebLOAD Programming Guide* *wlHttp*, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectedName (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The HTML name attribute value that the WebLOAD DOM has stored for the current object. Note that this is the *expected* name. WebLOAD uses the `IdentifyObject` method to verify whether or not this is the actual name value at run time.

Syntax

```
wlHttp.ExpectedName = "OptionalNameString"
```

Comment

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectedText (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The HTML text attribute value that the WebLOAD DOM has stored for the current object. Note that this is the *expected* text. WebLOAD uses the `IdentifyObject` method to verify whether or not this is the actual text value at run time.

Syntax

```
wlHttp.ExpectedText = "OptionalTextString"
```

Example

```
wlHttp.ExpectedText = "Pisces"
```

Comment

This property is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas.

See also

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ExpectNavigation() (method)

Method of Object

wlBrowser

Description

Stores information about the next user navigation activity, such as a `Navigate` or `Click` command. Initializes internal WebLOAD structures necessary before beginning a new navigation block.

`ExpectNavigation()` is also the starting point for each named navigation activity included in the Statistics Report, described in *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*. The navigation activity name is the second parameter of the `ExpectNavigation()` function. Each named navigation activity is identified in the Statistics Report by the name of the Web page to which the user navigated. If the user changes the page name, the navigation activity name also changes accordingly. A navigation activity is defined as all actions that appear between the `ExpectNavigation()` and the `SyncDOM()` commands.

Syntax

```
wlBrowser.ExpectNavigation("URL" [, "Title", "Target", "Method"])
```

Parameters

URL—String containing the URL of the original navigation saved at the time of recording.

[Title]—An optional string containing the title of the page returned during the original recording session. The page title is also used to identify this navigation activity by name when it appears in the Statistics Report. If the user changes the page name, the navigation activity name in the Statistics Report also changes accordingly.

[Target]—An optional string containing the destination frame of the navigation. This is either the top window or inner frame of the destination Web page. The Target parameter holds the same information as the `wlHttp.wlTarget` property, described on page 366.

Method—An optional string containing the name of the method to execute once the navigation to the specified window is complete. Usually `Get` or `Post`.

Return Value

None.

Example

```
wlBrowser.ExpectNavigation
    ("http://www.netizenbanking.com/cardApply.asp",
     "Apply For A Credit Card", "", "Get")
```

See also

Actions, page 20

Back(), page 27

Forward(), page 106

Refresh(), page 213

SyncDOM(), page 280

wlHttp, page 342

AutoNavigate(), page 26

ExpectNavigation(), page 97

Navigate(), page 186

SetWindow(), page 244

wlBrowser, page 325

wlTarget, page 366

File (object)

Property of Objects

`File` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Each `File` object represents one of the files referenced in a document. (Compare to the `element` object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the `form` object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["FILE"]
```

Methods

<code>wlClick()</code> , page 326	<code>wlMouseDown()</code> , page 347
<code>wlMouseOver()</code> , page 348	<code>wlMouseUp()</code> , page 349
<code>wlMultiSelect()</code> , page 350	<code>wlSelect()</code> , page 358
<code>wlTypeIn()</code> , page 366	

Properties

<code>id</code> , page 143	<code>Name</code> , page 184
<code>Size</code> , page 247	

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<code>Button</code> , page 34	<code>Checkbox</code> , page 38
<code>Collections</code> , page 47	<code>File</code> , page 99
<code>form</code> , page 102	<code>Image</code> , page 148
<code>InputButton</code> , page 157	<code>InputCheckbox</code> , page 158
<code>InputFile</code> , page 159	<code>InputImage</code> , page 160
<code>InputRadio</code> , page 162	<code>InputText</code> , page 164
<code>length</code> , page 167	<code>Radiobutton</code> , page 210
<code>Select</code> , page 231	<code>text</code> , page 291

TextArea, page 292*UIContainer*, page 301

File management functions

Description

These functions manage access to an Agenda's function and input files, including opening and closing files, copying files, specifying include files, and reading lines from ASCII input files.

Note that input file management is usually handled through the Form Data Wizard, described in *Using the Form Data Wizard*, page 35 in the *WebLOAD Programming Guide*. Output file management is also provided by the *wlOutputFile*.

See also

Close(), page 45*delete()*, page 70*GetLine()*, page 124*Open()*, page 193*Using the Form Data Wizard*, page 35 in the *WebLOAD Programming Guide**wlOutputFile*, page 351*Write()*, page 372*CopyFile()*, page 57*File management functions*, page 100*IncludeFile()*, page 150*Reset()*, page 219*Using the IntelliSense JavaScript Editor*, page 18*wlOutputFile()*, page 354*Writeln()*, page 373

fileName (property)

Mode

Working with this property through the *wlHttp.DataFile* is only done when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

*wlMediaPlayer**wlHttp.DataFile*

Description

This property is a string that holds the name of the file associated with the parent object.

If the parent is a *wlHttp.DataFile* object, then *FileName* holds the name of the file being submitted through an HTTP Post command.

If the parent is a `wlMediaPlayer` object, then `fileName` specifies or retrieves the name of the stream to play. (read/write string)

Syntax

wlHttp.Data:

When working with `wlHttp` objects, use the uppercase form:

<NA>

wlMediaPlayer

When working with `wlMediaPlayer` objects, use the lowercase form:

`MyMediaPlayerObject.fileName`

Comment

Notice that the `FileName` property for `wlHttp.DataFile` objects is written in uppercase.

See also

bitrate, page 30

currentPosition, page 62

Data, page 63

duration, page 80

fileName, page 100

Get(), page 110

OpenStream(), page 194

Play(), page 200

Resume(), page 220

Stop(), page 278

value, page 310

wlHttp, page 342

wlMediaPlayer(), page 345

connectionBandwidth, page 53

currentStreamName, page 62

DataFile, page 65

Erase, page 87

FormData, page 104

Header, page 138

Pause(), page 200

Post(), page 202

state, page 277

type, page 300

wlClear(), page 328

wlMediaPlayer, page 344

FindObject() (method)

Method of Object

wlBrowser

Description

A Dynamic Object Recognition (DOR) method that finds and returns a pointer to the next object now needed by the Agenda during playback. Objects include both simple objects, such as checkboxes or radio buttons, and complex objects, such as Java Applets, Flash, and ActiveX objects.

Syntax

```
wlBrowser.FindObject(ObjectType, ObjectIndex [, "ObjectName"])
```

Parameters

ObjectType—Type of object for which the method will search. Typical object types include FT_LINK, FT_FORM, and FT_OBJECT_ELEMENT.

ObjectIndex—Index number of object for which the method will search in the main collection.

ObjectName—Name of object for which the method will search, stored as a string. Optional.

Return Value

Pointer to the specified object, such as a form, link, or map, on the Web page.

Example

```
Form1 = wlBrowser.FindObject(FT_FORM, 1, "Query")
```

See also

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

Navigate(), page 186

SetWindow(), page 244

Dynamic Object Recognition (DOR) components, page 81

ObjectProperty[], page 189

wlBrowser, page 325

form (object)

Property of Object

form objects are grouped into collections of forms. The forms collection is a property of the following object:

document

Description

Specifies that the contained controls are all elements of a form. Each form object stores the parsed data for a complete HTML form (<FORM> tag). A form object contains the complete set of elements and input controls (text, radio buttons, checkboxes, etc.) that are all components

of a single form. (Compare to the *element* object, which stores the parsed data for a single HTML form element.)

`form` objects are local to a single thread. You cannot create new form objects using the JavaScript `new` operator, but you can access HTML forms through the properties and methods of the standard DOM objects. `form` properties are read-only.

`form` objects are grouped together within collections of `forms`, as described in *Collections*, page 47. A `forms` collection contains all form links (HTML `<FORM>` elements) within the document.

Syntax

The `forms` collection includes a `length` property that reports the number of `form` objects within a document (read-only). To find out how many `form` objects are contained within a document, check the value of:

```
document.forms.length
```

Use an index number to access an individual form's properties. Access each form's properties directly using the following syntax:

```
document.forms[index#].<form-property>
```

Note that you can also access a member of the `forms` collection by its HTML name attribute. For example, suppose that the first form on an HTML page is introduced by the tag:

```
<FORM name="SignUp"
      action="http://www.ABCDEF.com/FormProcessor.exe"
      method="post">
```

You can access this form by writing any of the following expressions:

```
document.forms[0]
document.forms["SignUp"]
document.forms.SignUp
document.SignUp
```

Methods

wlSubmit(), page 362

Properties

action, page 19

encoding, page 85

method, page 182

target, page 290

element, page 83

id, page 143

Name, page 184

Url, page 302

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Button</i> , page 34	<i>Checkbox</i> , page 38
<i>Collections</i> , page 47	<i>document</i> , page 79
<i>element</i> , page 83	<i>FindObject()</i> , page 102
<i>File</i> , page 99	<i>form</i> , page 102
<i>Image</i> , page 148	<i>InputButton</i> , page 157
<i>InputCheckbox</i> , page 158	<i>InputFile</i> , page 159
<i>InputImage</i> , page 160	<i>InputRadio</i> , page 162
<i>InputText</i> , page 164	<i>length</i> , page 167
<i>Radiobutton</i> , page 210	<i>Select</i> , page 231
<i>text</i> , page 291	<i>TextArea</i> , page 292
<i>window</i> , page 324	<i>wlSubmit()</i> , page 362

FormData (property)

Mode

This object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

A collection containing form field values. WebLOAD submits the field values to the HTTP server when you call the `Get()`, `Post()`, or `Head()` method of the `wlHttp` object.

The collection indices are the field names (HTML name attributes). Before you call `wlHttp.Post()`, set the value of each element to the data that you want to submit in the HTML field. The fields can be any HTML controls, such as buttons, text areas, or hidden controls.

Method

Use the *wlClear()* method, described on page 328, to delete specific FormData fields or clear all the FormData fields at once.

Comment

JavaScript supports two equivalent notations for named collection elements: `FormData.FirstName` or `FormData["FirstName"]`. The latter notation also supports spaces in the name, for example, `FormData["First Name"]`.

Getting FormData using Get()

You can get form data using a `Get ()` call. For example:

```
wlHttp.FormData["FirstName"] = "Bill"
wlHttp.FormData["LastName"] = "Smith"
wlHttp.FormData["EmailAddress"] = "bsmith@ABCDEF.com"
wlHttp.Get("http://www.ABCDEF.com/submit.cgi")
```

WebLOAD appends the form data to the URL as a query statement, using the following syntax:

```
http://www.ABCDEF.com/submit.cgi
?FirstName=Bill&LastName=Smith
&EmailAddress=bsmith@ABCDEF.com
```

Submitting FormData using Post()

Suppose you are testing an HTML form that requires name and email address data. You need to submit the form to the `submit.cgi` program, which processes the data. You can code this in the following way:

```
wlHttp.FormData["FirstName"] = "Bill"
wlHttp.FormData["LastName"] = "Smith"
wlHttp.FormData["EmailAddress"] = "bsmith@ABCDEF.com"
wlHttp.Post("http://www.ABCDEF.com/submit.cgi")
```

The `Post ()` call connects to `submit.cgi` and sends the `FormData` fields. In the above example, WebLOAD would post the following fields:

```
FirstName=Bill
LastName=Smith
EmailAddress=bsmith@ABCDEF.com
```

You may also submit `FormData` with missing fields or with data files. See *FormData*, page 227 in the *WebLOAD Programming Guide*, for more information.

See also

Data, page 63

Erase, page 87

FormData, page 104

DataFile, page 65

fileName, page 100

Get(), page 110

Header, page 138

type, page 300

wlClear(), page 328

Post(), page 202

value, page 310

wlHttp, page 342

Forward() (action)

Method of Object

wlBrowser

Description

Get and display the page for the next URL from the History list.

Syntax

```
wlBrowser.Forward()
```

Parameters

None.

Return Value

None

Example

<NA>

Comment

This method implements a special category of user action—clicking on a specific browser shortcut button from the Microsoft IE toolbar.

See also

Actions, page 20

Back(), page 27

Forward(), page 106

OnClick, page 190

Refresh(), page 213

wlClick(), page 326

wlBrowser, page 325

wlMouseOver(), page 348

wlMultiSelect(), page 350

AutoNavigate(), page 26

event, page 91

Navigate(), page 186

OnMouseOver, page 191

SetWindow(), page 244

wlClose(), page 329

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

wlSubmit(), page 362*wlTypeIn()*, page 366

frames (object)

Property of Object

The `frames` collection is a property of the following objects:

*document**window*

Description

The `frames` object retrieves a collection of all window objects defined by the given document or defined by the document associated with the given window. Each `window` object contains one of the child windows found in a browser window frameset. The `frames` collection stores the complete parse results for downloaded HTML frames, including nested child windows. Use the `frames` properties to retrieve information about any child windows to which the current window or document are linked.

`frames` collections are local to a single thread. WebLOAD creates an independent `frames` collection for each thread of an Agenda. You cannot create new `frames` collections using the JavaScript `new` operator, but you can access HTML frames through the properties and methods of the standard DOM objects. `frames` properties are read-only.

Syntax

The `frames` collection includes a `length` property that reports the number of `frame` objects within a document (read-only). To find out how many window objects are contained within a document, check the value of:

```
document.frames.length
```

Use an index number to access an individual frame's properties. Access each window's properties directly using the following syntax:

```
document.frames[#].<child-property>
```

Note that you can also access a member of the `frames` collection by its HTML name attribute. For example:

```
document.frames["namestring"]
```

OR

```
document.frames.namestring
```

Example

Access each window's properties directly through an index number:

```
document.frames[1].location
```

Access the first child window using the following expression:

```
frames[0]
```

Access the first child window's link objects directly using the following syntax:

```
frames[0].frames[0].links[#].<property>
```

For example:

```
document.frames[0].links[0].protocol
```

Properties

id, page 143

Index, page 151

length, page 167

Name, page 184

title, page 296

Url, page 302

window, page 324

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Collections, page 47

document, page 79

length, page 167

window, page 324

GeneratorName() (function)

Description

`GeneratorName()` provides a unique identification for the current Load Generator instance, even with multiple spawned processes running simultaneously. The identification string is composed of a combination of the current Load Generator name, computer name, and other internal markers.

Syntax

```
GeneratorName()
```

Parameters

None

Return Value

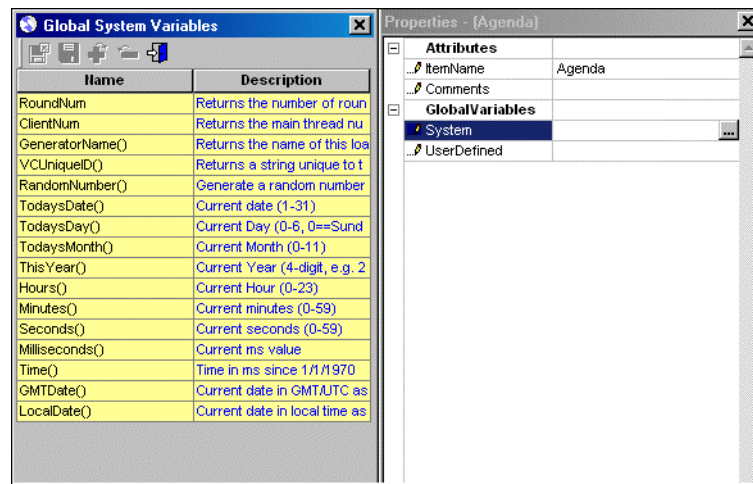
Returns a unique identification string for the current Load Generator.

Example

<NA>

GUI mode

WebLOAD recommends accessing global system variables, including the `GeneratorName()` identification function, through the Visual AAT GUI. To see a list of global system variables, highlight the top-level Agenda node in the Agenda Tree, find the `GlobalVariables` section of the Properties pane, and click on the browse button of the System field. A complete list of system variables and macros pops up, as illustrated in the following figure. The variables that appear in this list are available for use at any point in an Agenda file. For example, it is convenient to add `GeneratorName()` to a Message Node to clarify which Load Generator originated the messages that appear in the Console Log window.

**Figure 2-15: Global System Variables List****See also***ClientNum*, page 43*GetOperatingSystem()*, page 130*RoundNum*, page 221*GeneratorName()*, page 108*Identification variables and functions*, page 146*VUniqueID()*, page 312**Get() (method)**

Get() (addition method)

Method of Objects

This function is implemented as a method of the following objects:

wlGeneratorGlobal

wlSystemGlobal

Description

Returns the current value of the specified shared variable.

Syntax

```
Get ("SharedVarName", ScopeFlag)
```

Parameters

SharedVarName—The name of a shared variable whose value should be returned.

ScopeFlag—One of two flags, *WLCurrentAgenda* or *WLAllAgendas*, signifying the scope of the shared variable.

When used as a method of the *wlGeneratorGlobal* object:

- ◆ The *WLCurrentAgenda* scope flag signifies variable values that you wish to share between all threads of a single Agenda, part of a single process, running on a single Load Generator.
- ◆ The *WLAllAgendas* scope flag signifies variable values that you wish to share between all threads of one or more Agendas, common to a single spawned process, running on a single Load Generator.

When used as a method of the *wlSystemGlobal* object:

- ◆ The *WLCurrentAgenda* scope flag signifies variable values that you wish to share between all threads of a single Agenda, potentially shared by multiple processes, running on multiple Load Generators, system wide.
- ◆ The *WLAllAgendas* scope flag signifies variable values that you wish to share between all threads of all Agendas, run by all processes, on all Load Generators, system-wide.

Return Value

Returns the current value of the specified shared variable.

Example

```
CurrentCount =  
    wlGeneratorGlobal.Get ("MySharedCounter", WLCurrentAgenda)  
  
CurrentCount =  
    wlSystemGlobal.Get ("MyGlobalCounter", WLCurrentAgenda)
```


See also*Add()*, page 21*Set()*, page 237*wlGeneratorGlobal*, page 334*Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide**User-defined global properties*, page 304*wlSystemGlobal*, page 363**Get() (transaction method)**

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects

This function is implemented as a method of the following object:

*wlHttp***Description**

Perform an HTTP or HTTPS Get command. The method gets the `FormData`, `Data`, or `DataFile` properties in the Get command. In this way, you can get any type of data from an HTTP server.

Syntax

```
Get([URL] [, TransName])
```

Parameters

[URL]—An optional parameter identifying the document URL.

You may optionally specify the URL as a parameter of the method. `Get()` connects to the first URL that has been specified from the following list:

1. A `Url` parameter specified in the method call.
2. The `Url` property of the `wlHttp` object.
3. The local default `wlLocals.Url`.
4. The global default `wlGlobals.Url`.

[TransName]—An optional user-supplied string with the transaction name as it will appear in the Statistics Report, described in *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*.

Use *named transactions* to identify specific HTTP transactions by name. This simplifies assigning counters when you want WebLOAD to automatically calculate a specific transaction's occurrence, success, and failure rates.

The run-time statistics for transactions to which you have assigned a name appear in the Statistics Report. For your convenience, WebLOAD offers an Automatic Transaction option. On the Console, select Automatic Transaction from the General Tab of the Global Options dialog box. Automatic Transaction is set to True by default. With Automatic Transaction, WebLOAD automatically assigns a name to every Get and Post HTTP transaction. This makes statistical analysis simpler, since all HTTP transaction activity is measured, recorded, and reported for you automatically. You do not have to remember to add naming instructions to each Get and Post command in your Agenda. The name assigned by WebLOAD is simply the URL used by that Get or Post transaction. If your Agenda includes multiple transactions to the same URL, the information will be collected cumulatively for those transactions.

Return Value

None

Example

```
function InitAgenda()
    //Set the default URL
    wlGlobals.Url = "http://www.ABCDEF.com"
}

//Main script

//Connect to the default URL:
wlHttp.Get()

//Connect to a different, explicitly set URL:
wlHttp.Get("http://www.ABCDEF.com/product_info.html")

//Assign a name to the following HTTP transaction:
wlHttp.Get("http://www.ABCDEF.com/product_info.html",
           "UpdateBankAccount")
```

Use named transactions as a shortcut in place of the `BeginTransaction()...EndTransaction()` module. For example, this is one way to identify a logical transaction unit:

```
BeginTransaction("UpdateBankAccount")
    wlHttp.Get(url)
    ... // the body of the transaction
    ... // any valid JavaScript statements
    wlHttp.Post(url);
EndTransaction("UpdateBankAccount")
... // and so on
```

Using the named transaction syntax, you could write:

```

wlHttp.Get(url, "UpdateBankAccount")
... // the body of the transaction
... // any valid JavaScript statements
wlHttp.Post(url, "UpdateBankAccount")
... // and so on

```

For the HTTPS protocol, include "https://" in the URL and set the required properties of the `wlGlobals` object:

```
wlHttp.Get("https://www.ABCDEF.com")
```

Comment

You may not use the `TransName` parameter by itself. `Get()` expects to receive either *no* parameters, in which case it uses the Agenda's default URL, or *one* parameter, which must be an alternate URL value, or *two* parameters, including both a URL value and the transaction name to be assigned to this transaction.

See also

BeginTransaction(), page 28

CreateDOM(), page 59

CreateTable(), page 60

Data, page 63

DataFile, page 65

Erase, page 87

fileName, page 100

FormData, page 104

Header, page 138

ReportEvent(), page 214

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

VerificationFunction() (user-defined), page 320

wlHttp, page 342

GetCurrentValue() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHttp

Description

The method used by WebLOAD to identify the correct value for the specified field when working with dynamic HTML. This function is automatically added by ASM to your Agendas if necessary when recording with the Visual AAT.

Syntax

```
GetCurrentValue(FieldName, OrigValue)
```

Parameters

FieldName—The name of the field for which WebLOAD wishes to find a current value.

OrigValue—The expected field value, saved at recording time. If the value has not changed, this will be the value assigned to that field by `wlHttp.FormData`. If the value has changed, the new value will be identified by WebLOAD and used to replace the original field value.

Return Value

The current field value.

Example

```
wlHttp.FormData["FieldName"] =
    wlHttp.GetCurrentValue("FieldName", "OrigValue")
```

Comment

This method is part of the HTTP Protocol mode version of ASM, and is not used in most User Activity mode Agendas. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide* for more information.

See also

Automatic State Management for HTTP Protocol Mode, page 25

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

GetFieldValue() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the HTML value attribute (initial value) of a form field, given its name attribute.

Syntax

```
GetFieldValue(FieldName [, frame])
```

Parameters

`fieldName`—The name of the field whose value is to be retrieved.

`[frame]`—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested value of the specified field.

Example

```
ClientFirstName = wlHtml.GetFieldValue("FirstName")
```

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method only searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

GetFieldValueInForm() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the HTML value attribute (initial value) of a form field, given its name attribute. This method is similar to `GetFieldValue()`, but the search is limited to a specific form within a specific frame.

Syntax

```
GetFieldValueInForm(FormIndex, fieldName [, frame])
```

Parameters

`FormIndex`—Index number that identifies the specific form to which the search is to be limited.

`fieldName`—The name of the field whose value is to be retrieved.

[*frame*]
—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested HTML value attribute of the form field.

Example

If an HTML page includes two frames with a form in the second frame such as that illustrated in the Parse Tree example in *Parsing Web pages*, page 218 in the *WebLOAD Programming Guide*:

```
wlHtml.GetFieldValueInForm(0, "FirstName", Frame1)
searches the first form in Frame1 and returns "Bill".
```

Comment

The method does not search within nested frames. Omit the optional *frame* parameter if the HTML page does not contain frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetFormAction() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve a form object, representing a <FORM> element. The action attribute specifies the URL where the form data is to be submitted.

Syntax

```
GetFormAction(FormIndex [, frame])
```

Parameters

FormIndex—Index number that identifies the specific form to which the search is to be limited.

[*frame*]
—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested form object.

Example

If an HTML page includes two frames with a form in the second frame such as that illustrated in the Parse Tree example in *Parsing Web pages*, page 218 in the *WebLOAD Programming Guide*:

```
wlHtml.GetFormAction(0, Frame1)
```

returns a form object for the form.

Comment

The method does not search within nested frames. Omit the optional *frame* parameter if the HTML page does not contain frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetFrameByUrl() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve a frame object given its URL.

Syntax

```
GetFrameByUrl(UrlPattern [, frame])
```

Parameters

UrlPattern—The URL for the frame requested.

[*frame*]
—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested frame.

Example

Referring to the Parse Tree example in *Parsing Web pages*, page 218 in the *WebLOAD Programming Guide*:

```
//Retrieve Frame0
Frame0 = wlHtml.GetFrameByUrl("http://MyCompany/Frame0.html")

//Retrieve Frame0.1
Frame0_1 = wlHtml.GetFrameByUrl("http://MyCompany/Frame0B.html")
```

You may use * as a wildcard character in the URL. The method returns the first frame matching the search pattern. For example:

```
// To match URL (http://MyCompany/Frame0B.html)
Frame0_1 = wlHtml.GetFrameByUrl("*B.htm*")
```

You may narrow the search to frames nested within a specific parent frame by specifying the optional frame parameter. For example:

```
//Search within Frame0 and retrieve Frame0.0
Frame0_0 = wlHtml.GetFrameByUrl("*/MyCompany/*", Frame0)
```

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional *frame* parameter. In that case, the method searches within the specified *frame* and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetFrameUrl() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve a `location` object representing the URL of an HTML page. Optionally, specify a nested `frame`.

Syntax

```
GetFrameUrl ([frame])
```

Parameters

`[frame]`—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested location object.

Example

```
<NA>
```

Comment

This method is equivalent to the `location` property of a frame object (see *frames*, page 107).

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetHeaderValue() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the value of an HTTP header field.

Syntax

```
GetHeaderValue (HeaderName [, frame])
```

Parameters

`HeaderName`—The name of the header whose value is to be retrieved.

[*frame*]
—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested HTTP header field value.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

```
wlHtml.GetHeaderValue("Host")
returns "Server2.MyCompany.com".
```

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional *frame* parameter. In that case, the method searches within the specified *frame* and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetHost() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the host of a URL, including the port number.

Syntax

```
GetHost ([frame])
```

Parameters

[frame]—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested host information.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

```
wlHtml.GetHost ()
returns "Server2.MyCompany.com:80".
```

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetHostName() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the host name of a URL, not including the port number.

Syntax

```
GetHostName ([frame])
```

Parameters

[frame]—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested host name.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

```
wlHtml.GetHostName ()
returns "Server2.MyCompany.com".
```

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetIPAddress() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects*wlGlobals**wlLocals**wlHttp***Description**

Returns the identity of the *current* IP address.

Syntax

```
GetIPAddress ()
```

Parameters

None

Return Value

Returns a string with the IP address for the current Virtual Client.

Example

```
...
wlHttp.MultiIPSupport = "Yes"
CurrentIPAddress = wlHttp.GetIPAddress ()
wlHttp.Get ()
...
```

Comment

Requesting the identity of the *current* IP address is only meaningful if your Agenda is handling more than one IP address. `GetIPAddress ()` therefore can only return a value if `MultiIPSupport="Yes"`. If `MultiIPSupport` is turned off this method will return "Undefined".

The scope of `MultiIPSupport` depends, of course, on whether it was set through `wlGlobals`, `wlLocals`, or `wlHttp`. For example, if your Agenda sets `wlGlobals.MultiIPSupport`, then `GetIPAddress ()` returns a value at any point in the Agenda. If you set only `wlHttp.MultiIPSupport`, then `GetIPAddress ()` returns a value only if called before the next immediate HTTP transaction.

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlLocals, page 343

wlHttp, page 342

GetLine() (function)

Description

The `GetLine()` function reads and parses data from an ASCII file. The function reads the file one line at a time in the following way:

- ◆ If the user chose the `WLSequential` (default) mode, then:
 - The first `GetLine()` call in any thread of a Load Generator reads the first line of the file.
 - Each successive call in any thread of the Load Generator reads the next line of the file.
 - When the last line of the file has been read, the next access loops back to the first line of the file.
- ◆ If you decide to read the file entries in a random order, then each successive call in any thread of the Load Generator reads some randomly chosen line of the file. To read the input file lines in random order, you must include `Open(filename, WLRandom)` in the Agenda's `InitAgenda()` function.

In this way, a relatively small file can supply an unending stream of test data, and different clients are supplied with different sequences of data.

Syntax

```
GetLine(filename[, delimiter])
```

Parameters

`filename`—A string with the name of the file being read. May optionally include the full directory path.

`[delimiter]`—Optional character separating fields in one line of the input file. Default delimiter character is a comma.

Return Value

The `GetLine` function returns an array containing both the full lines and the individual tokens. The array (called `LineArray` in this example) includes the following elements:

- ◆ `LineArray[0]`—the complete line. For example:
"John,Smith, jsmith@ABC.com"
- ◆ `LineArray[1]`—the first token. In this example:
"John"
- ◆ `LineArray[2]`—the second token. In this example:
"Smith"
- ◆ `LineArray[3]`—the third token. In this example:

```
"jsmith@ABC.com"
```

- ◆ `LineArray.RoundNum`—number of rounds through the file (including the current round). For example: 4
- ◆ `LineArray.LineNum`—the number of the line that was just read. For example: 1

Example

To read and parse the next line of the `mydata.txt` ASCII input file, in this case including a directory path:

```
LineArray = GetLine("c:\\temp\\mydata.txt")
```

To specify a different delimiter:

```
LineArray = GetLine("c:\\temp\\mydata.txt", ":")
```

GUI mode

Note that the simplest way to work with input data from a file is through the Form Data Wizard, described in *Using the Form Data Wizard*, page 35 in the *WebLOAD Programming Guide*. For users who prefer to manually edit the JavaScript code in your Agenda and work directly with the `GetLine()` functions and `LineArray[]` commands, WebLOAD provides the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Select a `GetLine()` or `LineArray[]` command from the Edit | Insert | JavaScript | General menu. The Visual AAT automatically inserts the correct code for the selected command into the Agenda file. The user may then edit parameter values without any concerns about mistakes in the function syntax.

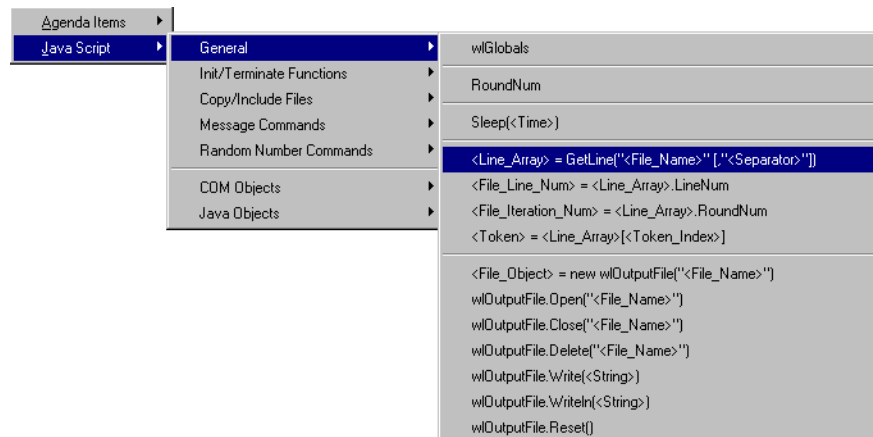


Figure 2-16: Inserting a `GetLine` function

Comment

JavaScript requires that you double the backslash in strings. If your directory path includes the backslash character, remember to double the backslashes, as in the preceding example.

If the line found in the file contains no separator characters, then the entire line is considered to be a single token. In that case, the function returns a two-element array (`LineArray[0]` and `LineArray[1]`), each containing the entire line.

See also

Close(), page 45

delete(), page 70

GetLine(), page 124

Open(), page 193

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

wlOutputFile(), page 351

Write(), page 372

CopyFile(), page 57

File management functions, page 100

IncludeFile(), page 150

Reset(), page 219

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile(), page 354

WriteLn(), page 373

GetLinkByName() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve a location object representing a link, given the hypertext display.

Syntax

```
GetLinkByName(Hypertext [, frame])
```

Parameters

Hypertext—The hypertext displayed in the desired link.

[*frame*]*—*An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested location object.

Example

Suppose the HTML on a page contains:


```
<A href="http://MyCompany/link1.html">Product information </A>
```

In this example,

```
wlHtml.GetLinkByName("Product information")
```

returns a location object for `http://MyCompany/link1.html`.

The search is case sensitive. You may use the `*` wildcard character in the Hypertext string. For example,

```
wlHtml.GetLinkByName("*roduct info*")
```

also returns an object for `http://MyCompany/link1.html`.

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetLinkByUrl() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve a location object representing a link, given part of the URL string.

Syntax

```
GetLinkByUrl(UrlPattern [, frame])
```

Parameters

UrlPattern—The URL of the desired link. Use the `*` wildcard character to represent the missing parts.

[frame]—An optional frame specification, used to limit the scope of the search to a specific frame.

Return Value

The requested location object.

Example

Suppose the HTML on a page contains:

```
<A href="http://MyCompany/link1.html">Product information </A>
```

In this example,

```
wlHtml.GetLinkByUrl ("*link1.htm*")
```

returns a location object for `http://MyCompany/link1.html`.

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetMessage() (method)

Method of Object

wlException

Description

Returns the message string text stored in this object.

Syntax

```
wlExceptionObject.GetMessage ()
```

Parameters

None

Return Value

Text string of the error message for this object.

Example

```
MeaningfulErrorMessage = myExceptionObject.GetMessage ()
```

See also

<i>Error Management</i> , page 104 in the <i>WebLOAD Programming Guide</i>	<i>ErrorMessage()</i> , page 89
<i>ExceptionEnabled</i> , page 92	<i>GetMessage()</i> , page 129
<i>GetSeverity()</i> , page 133	<i>InfoMessage()</i> , page 152
<i>LiveConnect Overview</i> , page 255 in the <i>WebLOAD Programming Guide</i>	<i>Message functions</i> , page 180
<i>ReportLog()</i> , page 215	<i>SevereErrorMessage()</i> , page 245
<i>Using the IntelliSense JavaScript Editor</i> , page 18	<i>WarningMessage()</i> , page 323
<i>wlException</i> , page 331	<i>wlException()</i> object constructor, page 333

GetOperatingSystem() (function)

Description

Returns a string identifying the operating system running on the current Load Generator.

Syntax

```
GetOperatingSystem()
```

Parameters

None

Return Value

Returns the name of the operating system running on the current Load Generator in the format of the operating system name followed by some version identification.

For example, if the Load Generator is working with a Solaris platform, this function would return the string 'Solaris' followed by the version name and release number, such as SunOS2.

If the Load Generator is working with a Linux platform, this function would return the string 'Linux' followed by the version name and release number, such as RedHat1.

If the Load Generator is working with a Windows platform, possible return values include:

- ◆ Windows 95
- ◆ Windows 98
- ◆ Windows NT/2000 (*ServicePack#*)
- ◆ Windows XP
- ◆ Windows (for any other Windows version)

Example

<NA>

See also*ClientNum*, page 43*GetOperatingSystem()*, page 130*RoundNum*, page 221*GeneratorName()*, page 108*Identification variables and functions*, page 146*VCUniqueID()*, page 312

GetPortNum() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object*wlHtml***Description**

Retrieve the port number of the current URL.

Syntax

```
GetPortNum([frame])
```

Parameters

[frame]—An optional frame specification, used to retrieve the port of a specific frame.

Return Value

The requested number.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

`wlHtml.GetPortNum()` would return a value such as 80.

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetQSFieldValue() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the value of a search attribute in a URL. The search attributes are the fields following the `?` symbol, appended to the end of a URL.

Syntax

```
GetQSFieldValue (Url, FieldName)
```

Parameters

`Url`—The complete URL string to be parsed and searched.

`FieldName`—The name of the field whose value is to be retrieved.

Return Value

The requested value.

Example

The following search string:

```
wlHtml.GetQSFieldValue ("http://www.ABCDEF.com/query.exe" +
    "?SearchFor=icebergs&SearchType=ExactTerm", "SearchFor")
```

returns "icebergs".

See also*wlHtml*, page 342*Parsing Web pages*, page 218 in the *WebLOAD Programming Guide*

GetSeverity() (method)

Method of Object*wlException***Description**

Returns the severity level value stored in this object.

Syntax*wlExceptionObject*.GetSeverity()**Parameters**

None

Return Value

Integer, representing one of the following error level values:

- ◆ *WLError*—this specific transaction failed and the current test round was aborted. The Agenda displays an error message in the Log window and begins a new round.
- ◆ *WLSevereError*—this specific transaction failed and the test session must be stopped completely. The Agenda displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

SeverityLevel = myExceptionObject.GetSeverity()

See also*Error Management*, page 104 in the *WebLOAD Programming Guide**ErrorMessage()*, page 89*ExceptionEnabled*, page 92*GetMessage()*, page 129*GetSeverity()*, page 133*InfoMessage()*, page 152*LiveConnect Overview*, page 255 in the *WebLOAD Programming Guide**Message functions*, page 180*ReportLog()*, page 215*SevereErrorMessage()*, page 245*Using the IntelliSense JavaScript Editor*, page 18*WarningMessage()*, page 323*wlException*, page 331*wlException()* object constructor, page 333

GetStatusLine() (method)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the status string from the HTTP header.

Syntax

```
GetStatusLine([frame])
```

Parameters

[frame]—An optional frame specification, used to retrieve the status string of a specific frame.

Return Value

The requested status string.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

`wlHtml.GetStatusLine()` would return "OK".

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetStatusNumber() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the status code from the HTTP header.

Syntax

```
GetStatusNumber ([frame])
```

Parameters

[frame]—An optional frame specification, used to retrieve the status code of a specific frame.

Return Value

The requested status number.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

`wlHtml.GetStatusNumber()` would return 200.

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

GetUri() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHtml

Description

Retrieve the URI part of a URL. The URI is the portion of the address following the host name.

Syntax

```
GetUri([frame])
```

Parameters

[frame]—An optional frame specification, used to retrieve the URI of a specific frame.

Return Value

The requested URI string.

Example

For the following HTTP Header example:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com:80
```

`wlHtml.GetUri()` would return `"WebPage.html"`.

Comment

By default, the method searches in all frames of the parse tree and returns the first match. You may narrow the search by specifying an optional `frame` parameter. In that case, the method searches within the specified `frame` and all its nested frames.

See also

wlHtml, page 342

Parsing Web pages, page 218 in the *WebLOAD Programming Guide*

hash (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

link

location

Description

The HTML anchor portion of the URL, not including the # initial symbol (read-only string).

Example

Given the following HTML fragment:

```
<A href="https://www.ABCDEF.com:80/products/order.html#modems">  
<A href="http://www.ABCDEF.com/search.exe?  
      SearchFor=modems&SearchType=ExactTerm">
```

`links[0].hash` is "modems".

See also

link, page 169

location, page 176

Head() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHttp, page 342

Description

Perform an HTTP or HTTPS Head command.

Syntax

`Head()`

Parameters

None

Return Value

None

Example

<NA>

Comment

This method operates in the same way as `Get ()`, but it retrieves only the HTTP or HTTPS header from the server. It does not download the body of the URL, such as a Web page.

See also

<i>Browser configuration components</i> , page 30	<i>Data</i> , page 63
<i>DataFile</i> , page 65	<i>FormData</i> , page 104
<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>	<i>wGlobals</i> , page 339
<i>wHttp</i> , page 342	<i>wLocals</i> , page 343
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

Header (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*wHttp***Description**

A collection of HTTP header fields that you want to send in a `Get ()`, `Post ()`, or `Head ()` call.

Example

By default, WebLOAD sends the following header in any HTTP command:

```
host: <host>
user-agent: Radview/HttpLoader 1.0
accept: */*
```

Here, <host> is the host name to which you are connecting, for example:

```
www.ABCDEF.com:81.
```

You may reset these properties, for example, as follows:

```
wlHttp.UserAgent = "Mozilla/4.03 [en] (WinNT; I) "
```

Alternatively, you can use the `Header` property to override one of the default header fields. For example, you can redefine the following header field:

```
wlHttp.Header["user-agent"] = "Mozilla/4.03 [en] (WinNT; I) "
```

GUI mode

WebLOAD offers a simple way to reset configuration properties using the various tabs of the Tools | Default Options dialog box. Resetting configuration properties as you run and rerun various testing scenarios allows you to fine tune your tests to match your exact needs at that moment.

For example, reset the user-agent value through the Browser Emulation tab, as illustrated in the following figure:

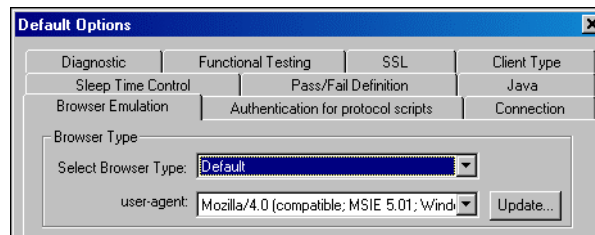


Figure 2-17: Setting user-agent value

Comment

Use the `wlClear()` method, described on page 328, to delete specific Header fields or clear all the Header fields at once.

You cannot override the host header or set a cookie header using the Header property. To set a cookie, see *wlCookie*, page 330

Use the `wlHttp.Header` property to change or reset specific individual values right before executing the next `wlHttp.ExpectNavigation()` command. Be careful—Note that any information set using the `wlHttp.Header` property *takes priority* over any defaults set through the GUI (recommended) or using the `wlGlobals`, `wlLocals`, or `wlHttp` properties. If there is any discrepancy between the document header information and the HTTP values, WebLOAD will work with the information found in the `wlHttp.Header` property while also issuing a warning to the user.

See also

<i>Browser configuration components</i> , page 30	<i>Data</i> , page 63
<i>DataFile</i> , page 65	<i>Erase</i> , page 87
<i>fileName</i> , page 100	<i>FormData</i> , page 104
<i>Get()</i> , page 110	<i>Header</i> , page 138
<i>Post()</i> , page 202	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>type</i> , page 300	<i>UserAgent</i> , page 306
<i>value</i> , page 310	<i>wlClear()</i> , page 328
<i>wlGlobals</i> , page 339	<i>wlHeader</i> , page 340
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

HistoryLimit (property)

Property of Objects*wlGlobals***Description**

Sets a limit to the number of commands to be saved in a history file when working with the `UseHistory` property.

Example

<NA>

UseHistory, page 304*wlGlobals*, page 339

host (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*link**location*

Description

The host portion of the URL, including both the host name and the port (read-only string).

Example

Given the following HTML fragment:

```
<A href="https://www.ABCDEF.com:80/products/order.html#modems">
<A href="http://www.ABCDEF.com/search.exe?
      SearchFor=modems&SearchType=ExactTerm">
```

links[0].host is "www.ABCDEF.com:80"

See also

link, page 169

location, page 176

hostname (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

link

location

Description

The host name portion of the URL (read-only string).

Example

Given the following HTML fragment:

```
<A href="https://www.ABCDEF.com:80/products/order.html#modems">
<A href="http://www.ABCDEF.com/search.exe?
      SearchFor=modems&SearchType=ExactTerm">
```

links[0].hostname is "www.ABCDEF.com"

See also

link, page 169

location, page 176

href (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

link

location

Description

The complete URL of the link (read-only string).

Example

Given the following HTML fragment:

```
<A href="https://www.ABCDEF.com:80/products/order.html#modems">
<A href="http://www.ABCDEF.com/search.exe?
      SearchFor=modems&SearchType=ExactTerm">
```

```
links[0].href is
"https://www.ABCDEF.com/products/order.html#modems"
```

Comment

Note that the `href` property contains the entire URL. The other `link` properties contain portions of the URL. `links[#].href` is the default property for the `link` object. For example, if

```
links[0]='http://microsoft.com'
```

then the following two URL specifications are equivalent:

```
mylink=links[0].href
```

and

```
mylink=links[0]
```

See also

link, page 169

location, page 176

httpEquiv (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlMeta

Description

Retrieves the value of the HTTP-EQUIV attribute of the META tag (read-only string).

Syntax

```
wlMetas[index#].httpEquiv
```

Example

```
document.wlMetas[0].httpEquiv
```

See also

content, page 56

Name, page 184

wlMeta, page 346

httpEquiv, page 143

Url, page 302

id (property)

Property of Objects

area

Div

form

Image

InputCheckbox

InputImage

InputPassword

link

map

Select

Button

element

frames

InputButton

InputFile

InputRadio

InputText

location

script

Span

<i>table</i>	<i>TableCell</i>
<i>TextArea</i>	<i>UIContainer</i>
<i>wlTable</i>	<i>wlXmIs</i>

Description

Retrieves the string identifying the parent object. The ID value is taken from the ID attribute within the tag. This property is optional. If this object does not have an ID attribute then the value is undefined.

When working with `element`, `forms`, `frames`, `image`, or `map` objects, returns a string containing an alternative identification means for the complete image, map, forms or frame or for elements of type Button, CheckBox, File, Image, Password, Radio, Reset, Select, Submit, Text, and TextArea.

Example

wlTables example:

If the first table on a page is assigned the ID tag `myTable`, access the table using any of the following:

```
document.wlTables[0]
or
document.wlTables.myTable
or
document.wlTables[myTable]
```

If duplicate identifiers are found, the `id` property will refer to the first `wlTable` object found with that identifier.

wlXmIs example:

If the first XML object on a page is assigned the ID tag `myXmlDoc`, access the object using any of the following:

```
MyBookstore = document.wlXmIs[0]
or
MyBookstore = document.wlXmIs.myXmlDoc
or
MyBookstore = document.wlXmIs["myXmlDoc"]
```

If duplicate identifiers are found, the `id` property will refer to the first XML object found with that identifier.

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

area, page 24

cell, page 35 (*wlTable* and *row* property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

Div, page 77

form, page 102

id, page 143 (*wlTable* property)

InnerHTML, page 153 (*cell* property)

InputButton, page 157

InputFile, page 159

InputPassword, page 161

InputText, page 164

load(), page 170

load() and *loadXML()* method comparison, page 172

map, page 178

Prepare(), page 205

row, page 223 (*wlTable* property)

script, page 229

Span, page 254

table, page 284

TableCompare, page 286

TextArea, page 292

wlTable, page 364

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Button, page 34

cellIndex, page 37 (*cell* property)

cols, page 48 (*wlTable* property)

CompareColumns, page 51

Details, page 72

element, page 83

frames, page 107

Image, page 148

InnerText, page 155 (*cell* property)

InputCheckbox, page 158

InputImage, page 160

InputRadio, page 162

link, page 169

loadXML(), page 175

location, page 176

MatchBy, page 178

ReportUnexpectedRows, page 216

rowIndex, page 225 (*row* property)

Select, page 231

src, page 255

TableCell, page 285

tagName, page 289 (*cell* property)

UIContainer, page 301

wlXmIs, page 370

XMLDocument, page 380

Identification variables and functions

Description

For performance statistics to be meaningful, testers must be able to identify the exact point being measured. WebLOAD therefore provides the following identification variables and functions:

- ◆ Two variables, `ClientNum` and `RoundNum`, identify the client and round number of the current Agenda instance.
- ◆ The `GeneratorName()` function identifies the current Load Generator.
- ◆ The `GetOperatingSystem()` function identifies the operating system of the current Load Generator.
- ◆ The `VCUniqueID()` function identifies the current Virtual Client instance.

Example

The following example illustrates common use of these variables and functions. Use these variables and function to support the WebLOAD measurement features and obtain meaningful performance statistics.

Suppose your Agenda submits data to a server on an HTML form. You want to label one of the form fields so you can tell which WebLOAD client submitted the data, and in which round of the main script.

You can do this using a combination of the `ClientNum` and `RoundNum` variables. Together, these variables uniquely identify the WebLOAD client and round. For example, you can submit a string such as the following in a form field:

```
"C" + ClientNum.toString() + "R" + RoundNum.toString()
```

GUI mode

WebLOAD recommends accessing These identification variables and functions through the Visual AAT GUI, illustrated here. To see a list of global system variables, highlight the top-level Agenda node in the Agenda Tree, find the `GlobalVariables` section of the Properties pane, and click on the browse button of the System field. The following list appears. All the variables that appear in this list are available for use at all times in an Agenda file. For example, it is convenient to add `ClientNum` to a Message Node to clarify which client sent the messages that appear in the Console Log window.

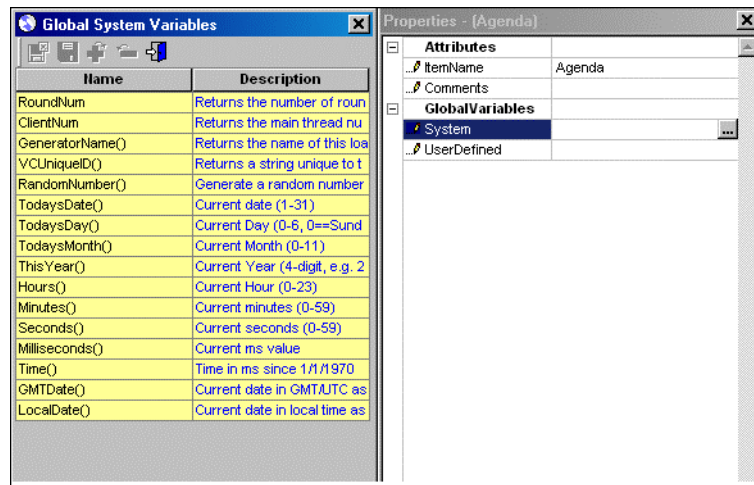


Figure 2-18: Global System Variables List

See also*ClientNum*, page 43*GeneratorName()*, page 108*GetOperatingSystem()*, page 130*Identification variables and functions*, page 146*RoundNum*, page 221*VCUniqueID()*, page 312

IdentifyObject() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object*wlHttp***Description**

The method used by WebLOAD to identify the correct URL for the current target frame when working with dynamic HTML. This function is part of the HTTP Protocol set of ASM functions.

Syntax

```
IdentifyObject (URLString)
```

Parameters

`URLString`—The expected URL value, saved at recording time. If the URL has not changed, this will be the value assigned to `wlHttp.Url`. If the URL has changed, the new value will be identified by WebLOAD and used to replace the original URL value.

Return Value

The current URL value.

Example

```
wlHttp.Url = wlHttp.IdentifyObject("URLstring")
```

See also

Automatic State Management for HTTP Protocol Mode, page 25 *Browser configuration components*, page 30

Dynamic Object Recognition (DOR) components, page 81 *wlHttp*, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Image (object)

Description

Each `Image` object represents one of the images or video clips embedded in a document (HTML `` element). `Image` objects are accessed through `Images` Collections. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

`image` objects are grouped together within collections of `images`, accessed either directly through the `document` object (`document.images[#]`), or through the `document.all` collection.

Syntax**Recommended:**

```
document.all.tags["IMG"]
```

Alternative (HTTP mode):

To find out how many `image` objects are contained within a document, check the value of:

```
document.images.length
```

Access each image's properties directly using the following syntax:

```
document.images[index#].<image-property>
```

Example**Recommended:**

```
document.all.tags["IMG"]
```

Alternative (HTTP mode):

```
document.images[1].src
```

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

Alt, page 23

InnerLink, page 155

OuterLink, page 196

src, page 255

id, page 143

Name, page 184

protocol, page 208

Url, page 302

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

File, page 99

Image, page 148

InputButton, page 157

InputFile, page 159

InputRadio, page 162

Radiobutton, page 210

text, page 291

UIContainer, page 301

Checkbox, page 38

document, page 79

form, page 102

length, page 167

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Select, page 231

TextArea, page 292

IncludeFile() (function)

Description

Instructs WebLOAD to include the specified file, and optionally execute scripts that are stored within that file, as part of the initialization process before beginning the main Agenda execution rounds. Encourages modular programming by enabling easy access to sets of library function files.

Syntax

```
IncludeFile(filename[, WLExecuteScript])
```

Parameters

`filename`—A string or variable containing the full literal name of the file to be included. Specific system directories are searched for the specified file in the search order described in the preceding section. Once the file is found, any functions or variables defined within that file are compiled and included within the calling Agenda when the Agenda is compiled.

`WLExecuteScript`—`WLExecuteScript` is a global constant that acts as a flag when passed as a parameter to `IncludeFile()`. `WLExecuteScript` is an optional parameter. When included, WebLOAD will not only compile the definitions found in the specified file. WebLOAD will also execute any additional commands or functions found within that file outside the included function definitions. With `WLExecuteScript`, WebLOAD enables work with self-initializing include files that can define, set, and execute the commands necessary to initialize a work environment at compile time.

Return Value

None

Example

To include the external file `MyFunction.js`, located on the Console during WebLOAD testing, use the following command:

```
function InitAgenda() {  
    IncludeFile("MyFunction.js")  
}
```

GUI mode

Note that `CopyFile()` and `IncludeFile()` functions can be added directly to the code in an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Select a function from the Edit | Insert | JavaScript | Copy/Include Files menu. The Visual AAT automatically inserts the correct code for the selected function into the Agenda file. The user may then edit parameter values without any concerns about mistakes in the function syntax.

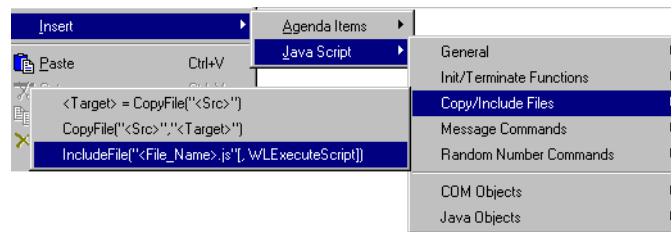


Figure 2-19: Inserting an IncludeFile function

Comment

The `IncludeFile` command must be inserted in the `InitAgenda()` section of your JavaScript program.

See also

Close(), page 45

delete(), page 70

GetLine(), page 124

Open(), page 193

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

wlOutputFile, page 351

Write(), page 372

CopyFile(), page 57

File management functions, page 100

IncludeFile(), page 150

Reset(), page 219

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile(), page 354

Writeln(), page 373

Index (property)

Property of Objects

frames

Description

Sets or retrieves the index number of the parent object. For example, the ordinal position of an option in a list box.

Syntax

<NA>

See also

Collections, page 47

frames, page 107

length, page 167

InfoMessage() (function)

Description

Displays a generally informative (but not necessarily problematic) message in the Log Window.

Syntax

```
InfoMessage (msg)
```

Parameters

`msg`—A string with an informative message to be sent to the Console.

Return Value

None.

Example

<NA>

Comment

If you call `InfoMessage ()` in the main script, WebLOAD sends an informative message to the Log window and continues with Agenda execution as usual. The message has no impact on the continued execution of the WebLOAD test.

GUI mode

WebLOAD recommends adding message functions to your Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a Message Node to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

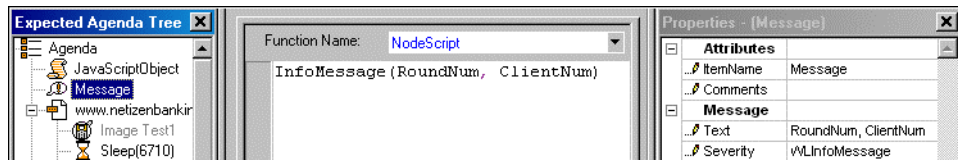


Figure 2-20: Adding a Message Node to an Agenda

See also

Error Management, page 104 in the *WebLOAD Programming Guide* *ErrorMessage()*, page 89

ExceptionEnabled, page 92 *GetMessage()*, page 129

GetSeverity(), page 133 *InfoMessage()*, page 152

LiveConnect Overview, page 255 in the
WebLOAD Programming Guide

ReportLog(), page 215

Using the IntelliSense JavaScript Editor, page 18

wlException, page 331

Message functions, page 180

SevereErrorMessage(), page 245

WarningMessage(), page 323

wlException() object constructor, page 333

InnerHTML (property)

Property of Objects

cell

wlXmIs

script

Description

Sets or retrieves the HTML found between the start and end tags of the object.

Syntax

When working with *cell* objects, use the uppercase form:

```
...cells[2].InnerHTML
```

When working with *script* or *wlXmIs* objects, use the lowercase form:

```
...scripts[2].innerHTML
```

Example

```
<NA>
```

Comment

Notice that the *InnerHTML* property for *cell* objects is written in uppercase.

WebLOAD recommends managing scripts, tables, and XML DOM objects on a Web page through the standard *document.all* collection.

See also

cell, page 35 (*wlTable* and *row* property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

id, page 143 (*wlTable* and *wlXmIs* property)

InnerImage, page 154

cellIndex, page 37 (*cell* property)

cols, page 48 (*wlTable* property)

CompareColumns, page 51

Details, page 72

InnerHTML, page 153 (*cell* and *wlXmIs* property)

InnerText, page 155 (*cell* property)

<i>load()</i> , page 170	<i>loadXML()</i> , page 175
<i>load() and loadXML() method comparison</i> , page 172	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<i>wlTable</i> property)	<i>RowIndex</i> , page 225 (<i>row</i> property)
<i>script</i> , page 229	<i>src</i> , page 255
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<i>cell</i> property)
<i>wlTable</i> , page 364	<i>wlXmIs</i> , page 370
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>XMLDocument</i> , page 380

InnerImage (property)

Property of Object

<i>Button</i>	<i>element</i>
<i>link</i>	<i>location</i>
<i>TableCell</i>	<i>UIContainer</i>

Description

Sets or retrieves the image found between the <Start> and <End>tags of the object. When working with a *button* object, the image that appears on the button. When working with a *link* or *location* object, the image that appears over the link. When working with a *TableCell* object, the image that appears over a table cell.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Collections</i> , page 47	<i>id</i> , page 143
<i>InnerHTML</i> , page 153	<i>InnerImage</i> , page 154
<i>InnerText</i> , page 155	<i>src</i> , page 255

InnerLink (property)

Property of Objects

Image

Description

Represents the inner link field for the parent `image` object.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

InnerText (property)

Property of Object

Button

Div

link

Span

UIContainer

cell

element

location

TableCell

Description

Sets or retrieves *only the text* found between the <Start> and <End>tags of the object. When working with a `Button` element object, the text that appears on the button. When working with a `link` or `location` object, the text that appears over the link. When working with a `TableCell` object, the text that appears over a table cell.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34
cell, page 35 (`wlTable` and `row` property)
Collections, page 47
Compare(), page 49
CompareRows, page 52
Div, page 77
id, page 143 (`wlTable` and `wlXmIs` property)
InnerImage, page 154
link, page 169
MatchBy, page 178
ReportUnexpectedRows, page 216
rowIndex, page 225 (`row` property)
src, page 255
TableCompare, page 286
UIContainer, page 301
Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*
cellIndex, page 37 (`cell` property)
cols, page 48 (`wlTable` property)
CompareColumns, page 51
Details, page 72
element, page 83
InnerHTML, page 153 (`cell` and `wlXmIs` property)
InnerText, page 155 (`cell` property)
location, page 176
Prepare(), page 205
row, page 223 (`wlTable` property)
Span, page 254
TableCell, page 285
tagName, page 289 (`cell` property)
wlTable, page 364

InputButton (object)

Property of Objects

`InputButton` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = button`. Each `InputButton` object represents one of the input buttons embedded in a form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["BUTTON"]
```

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

wlMultiSelect(), page 350

wlSelect(), page 358

Properties

event, page 91

id, page 143

InnerText, page 155

Name, page 184

OnClick, page 190

title, page 296

value, page 310

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Checkbox, page 38

Collections, page 47

File, page 99

form, page 102

Image, page 148

InputButton, page 157

InputCheckbox, page 158

InputFile, page 159

InputImage, page 160

InputRadio, page 162

InputText, page 164

length, page 167*Radiobutton*, page 210*Select*, page 231*text*, page 291*TextArea*, page 292*UIContainer*, page 301

InputCheckbox (object)

Property of Objects

`InputCheckbox` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = checkbox`. Each `InputCheckbox` object represents one of the input checkboxes embedded in a form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["CHECKBOX"]
```

Methods

wlClick(), page 326*wlMouseDown()*, page 347*wlMouseOver()*, page 348*wlMouseUp()*, page 349*wlMultiSelect()*, page 350*wlSelect()*, page 358

Properties

AdjacentText, page 23*id*, page 143*Name*, page 184*title*, page 296*value*, page 310

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99*form*, page 102*Image*, page 148

<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>length</i> , page 167	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301

InputFile (object)

Property of Objects

`InputFile` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = file`. Each `InputFile` object represents a file upload object with a text box and Browse button. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["FILE"]
```

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlTypeIn()</i> , page 366	

Properties

<i>id</i> , page 143	<i>Name</i> , page 184
<i>Size</i> , page 247	

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Button</i> , page 34	<i>Checkbox</i> , page 38
<i>Collections</i> , page 47	<i>File</i> , page 99
<i>form</i> , page 102	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>length</i> , page 167	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301

InputImage (object)

Property of Objects

`InputImage` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = image`. Each `InputImage` object represents an embedded image control that, when clicked, causes the form to be immediately submitted. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["IMG"]
```

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358

Properties

<i>Alt</i> , page 23	<i>id</i> , page 143
<i>Name</i> , page 184	<i>Url</i> , page 302

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Button</i> , page 34	<i>Checkbox</i> , page 38
<i>Collections</i> , page 47	<i>File</i> , page 99
<i>form</i> , page 102	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>length</i> , page 167	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301

InputPassword (object)

Property of Objects

InputPassword objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = password`. Each InputPassword object represents a single-line text entry control similar to the `INPUT type=text` control, except that text is not displayed as the user enters it. This is used to represent an input password text field embedded in a form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["PASSWORD"]
```

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358

wlTypeIn(), page 366

Properties

id, page 143

Name, page 184

Size, page 247

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Checkbox, page 38

Collections, page 47

File, page 99

form, page 102

Image, page 148

InputButton, page 157

InputCheckbox, page 158

InputFile, page 159

InputImage, page 160

InputRadio, page 162

InputText, page 164

length, page 167

Radiobutton, page 210

Select, page 231

text, page 291

TextArea, page 292

UIContainer, page 301

InputRadio (object)

Property of Objects

InputRadio objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = radio`. Each InputRadio object represents one of the input radiobuttons embedded in a form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["RADIO"]
```

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358

Properties

<i>AdjacentText</i> , page 23	<i>id</i> , page 143
<i>Name</i> , page 184	<i>value</i> , page 310

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Button</i> , page 34	<i>Checkbox</i> , page 38
<i>Collections</i> , page 47	<i>File</i> , page 99
<i>form</i> , page 102	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>length</i> , page 167	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301

InputText (object)

Property of Objects

`InputText` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = text`. Each `InputText` object represents a single-line text entry control embedded in a form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["TEXT"]
```

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlTypeIn(), page 366

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

id, page 143

Size, page 247

Name, page 184

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

KeepAlive (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects*wlGlobals**wlHttp**wlLocals***Description**

Enable WebLOAD to keep an HTTP connection alive between successive accesses in the same round of the main script. The possible values are:

- ◆ No—Do not keep an HTTP connection alive.
- ◆ Yes—Keep the connection alive if the server permits.
(default)

Keeping a connection alive saves time between accesses. WebLOAD attempts to keep the connection alive unless you switch to a different server. However, some HTTP servers may refuse to keep a connection alive.

Use the `wlHttp.CloseConnection()` method to explicitly close a connection that you have kept alive. Otherwise, the connection is automatically closed at the end of each round.

Example

<NA>

Comment

You should not keep a connection alive if establishing the connection is part of the performance test.

GUI mode

WebLOAD recommends maintaining or closing connections through the Console GUI. Enable maintaining connections for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

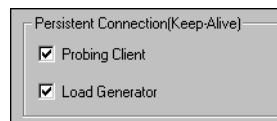


Figure 2-21: Enabling Persistent Connections for Load Generator

See also

Browser configuration components, page 30

CloseConnection(), page 46

Rules of scope for local and global variables,
page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

key (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

Header *wlHeader*
wlSearchPair

Description

The search key name (read-only).

Syntax

For wlHeaders:

```
document.wlHeaders[index#].key = "TextString"
```

For wlSearchPairs:

```
document.links[1].wlSearchPairs[index#].key = "TextString"
```

For wlHttp.Header:

```
wlHttp.Header["key"] = "TextString"
```

Example

For wlHeaders:

```
document.wlHeaders[0].key = "Server"
```

For wlSearchPairs:

```
document.links[1].wlSearchPairs[0].key = "Server"
```

For wlHttp.Header:

```
wlHttp.Header["key"] = "Server"
```

See also

Header, page 138

value, page 310

wlHeader, page 340

wlSearchPair, page 356

language (property)

Property of Object

script

Description

Retrieves the language in which the current script is written.

Example

"javascript" specifies that the script is written in JavaScript.

"vbscript" specifies that the script is written in Visual Basic Script.

See also

script, page 229

length (property)

Property of Object

All objects stored in *Collections*

form

frames

Description

Each collection of objects includes the single property `length`, which contains the size of the collection, that is, the number of objects included in this collection. Use an *Index* value to access the individual objects, and their properties, from within a collection.

Syntax

`collection.length`

Example

Frames collection

When accessing a `frames` collection, this property holds the size of the `frames` collection, that is, the number of nested window objects (read-only). To find out how many window objects are contained within this collection, check the value of:

```
...frames.length
```

If you were looking at a collection of frames within a root window, you would check the value of:

```
frames.length
```


If you were looking at a collection of frames within a document under an [implicit] root window, you would check the value of:

```
document.frames.length
```

Access each child window's properties directly using the following syntax:

```
document.frames[index#].<child-property>
```

For example:

```
document.frames[1].location
```

Forms collection

To find out how many `form` objects are contained within the `forms` collection of a document, check the `length` value:

```
NumberOfItems = document.forms.length
```

To access each individual form's properties directly with an index number, use the following syntax:

```
document.forms[1].action
```

See also

Collections, page 47

form, page 102

frames, page 107

Index, page 151

link (object)

Property of Objects

When working in the HTTP Protocol mode, links on a Web page are accessed through `link` objects that are grouped into collections of `links`. The `links` collection is a property of the document object.

When working in the User Activity mode, links on a Web page are accessed through `link` objects that are accessed through the `document.all` collection.

Description

A `link` object contains information on an external document to which the current document is linked. Each `link` object points to one of the URL links (HTML `<A>` elements) within the document. Each `link` object stores the parsed data for the HTML link (`<A>` element).

`link` objects are local to a single thread. You cannot create new `link` objects using the JavaScript `new` operator, but you can access HTML links through the properties and methods of the standard DOM objects. `link` properties are read-only.

link objects are organized into *Collections* of links or anchors. To access an individual link's properties, check the `length` property of the `links` collection and use an index number to access the individual links.

Syntax

HTTP Protocol mode:

To find out how many link objects are contained within a document, check the value of:

```
document.links.length
```

Access each link's properties directly using the following syntax:

```
document.links[#].<link-property>
```

User Activity mode:

```
<NA>
```

Example

HTTP Protocol mode:

```
document.links[1].protocol
```

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

Properties

id, page 143

InnerImage, page 154

InnerText, page 155

Name, page 184

OnClick, page 190

title, page 296

Url, page 302

When working in HTTP Protocol mode, links include the following properties:

hash, page 137

host, page 141

hostname, page 141

href, page 142

id, page 143

InnerText, page 155

Name, page 184

pathname, page 199

port, page 202

protocol, page 208

search, page 230

target, page 290

title, page 296

Url, page 302

wlSearchPair, page 356

Comment

WebLOAD recommends managing links on a Web page through the standard `document.all` collection of `link` objects, rather than through the HTTP Protocol mode `links` collection.

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Collections, page 47

document, page 79

FindObject(), page 102

length, page 167

load() (method)

Mode

The `load()` method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects

When working in the HTTP Protocol mode, XML DOM objects on a Web page are accessed through collections of `wlXmIs` objects. The `load()` function is a method of the following object:

`wlXmIs`

Description

Call `load(URL)` to download XML documents from a Web site and automatically load these documents into XML DOM objects.

Do not include any external references when using `load()`.

`load()` relies on the MSXML parser to perform any HTTP transactions needed to download the XML document. The MSXML module accesses external servers and completes all necessary transactions without any control or even knowledge on the part of the WebLOAD system tester. From WebLOAD's perspective, these transactions are never performed in the context of the test session. For this reason, any settings that the user enters through the WebLOAD Agenda or Console will not be relayed to the MSXML module and will have no effect on the document 'load'. For the same reason, the results of any transactions completed this way will not be included in the WebLOAD statistics reports.

Syntax

```
load(URLString)
```

Parameters

`URLString`—String parameter with the URL or filename where the XML document may be found.

Return Value

None.

Example

```
myXMLDoc = document.wlXmIs[0]
myXMLDoc.load("http://server/xmIs/file.xml")
```

Comment

You may use `load()` repeatedly to load and reload XML data into XML DOM objects. Remember that each new 'load' into an XML DOM object will overwrite any earlier data stored in that object.

WebLOAD recommends managing XML DOM objects on a Web page through the standard `document.all` collection rather than using the `wlXmIs` family of objects.

See also

Collections, page 47

id, page 143

InnerHTML, page 153

load(), page 170

loadXML(), page 175

load() and loadXML() method comparison, page 172

src, page 255

wlXmIs, page 370

XMLDocument, page 380

load() and loadXML() method comparison

Description

WebLOAD supports both the `load()` and the `loadXML()` methods to provide the user with maximum flexibility. The following table summarizes the advantages and disadvantages of each method:

Table 2-2: load() and loadXML() comparison

	Advantages	Disadvantages
loadXML()	Parameters that the user has defined through WebLOAD for the testing session will be	The method fails if the DTD section of the XML document string includes any external references.

	Advantages	Disadvantages
	applied to this transaction.	
load()	The user may load XML files that include external references in the DTD section.	<p>Parameters that the user has defined through WebLOAD for the testing session will not be applied to this transaction.</p> <p>WebLOAD does not record the HTTP Get operation. (See note below).</p> <p>The transaction results are not included in the session statistics report.</p> <p>Using this method may adversely affect the test session results.</p>

Comment

If you wish to measure the time it took to load the XML document using the `load()` method, create a timer whose results will appear in the WebLOAD statistics. For example:

```
myXMLDoc = document.wlXmIs[0]
SetTimer("GetXMLTime")
myXMLdoc.load("http://server/xmls/file.xml")
SendTimer("GetXMLTime")
```

See also

Collections, page 47

InnerHTML, page 153

loadXML(), page 175

src, page 255

XMLDocument, page 380

id, page 143

load(), page 170

load() and loadXML() method comparison, page 172

wlXmIs, page 370

LoadGeneratorThreads (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

wlGlobals

wlHttp

wlLocals

Description

Optionally, WebLOAD can allocate extra threads to download nested images and frames.

For clients that you define in a Load Generator, this option is controlled by the `LoadGeneratorThreads` property. The default value of this property is "Single", which means that Virtual Clients will not use extra threads to download data from the Server.

For the Probing Client, the option is controlled by the `ProbingClientThreads` property. The default is "Multiple", which means that the client can use three extra threads for nested downloads. This simulates the behavior of Web browsers, which often use extra threads to download nested images and frames.

The possible values of these properties are:

- ◆ `Single`—Do not use extra threads to download nested images and frames. (default for `LoadGeneratorThreads`)
- ◆ `Multiple`—Allocate three extra threads per client (for a total of four threads per client) to download nested images and frames. (default for `ProbingClientThreads`)
- ◆ Any specific number of threads between 1 and 8, such as "5"—Allocate that exact number of extra threads per client to download nested images and frames.

Example

You can assign any of these properties independently within a single Agenda. In that case, if you configure a Probing Client to run the Agenda, WebLOAD uses the value of `ProbingClientThreads` and ignores `LoadGeneratorThreads` (vice versa if you configure a Load Generator to run the Agenda). For example, you might write:

```
function InitAgenda() {
    //Do not use extra threads if a
    // Probing Client runs the Agenda
    wlGlobals.ProbingClientThreads = "Single"

    //Use extra threads if a
    // Load Generator runs the Agenda
    wlGlobals.LoadGeneratorThreads = "Multiple"
}
```

Comment

The extra threads have no effect on the `ClientNum` value of the client. The `ClientNum` variable reports only the main thread number of each client, not the extra threads.

GUI mode

WebLOAD recommends enabling or disabling multi-threaded virtual clients through the Console GUI. Enable multi-threading for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box and setting the number of threads you prefer, as illustrated in the following figure:

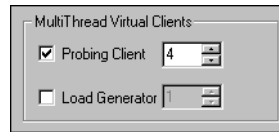


Figure 2-22: Enabling multi-threading for Load Generator

See also

Browser configuration components, page 30

ProbingClientThreads, page 206

Rules of scope for local and global variables,
page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

loadXML() (method)

Mode

The `loadXML()` method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

When working in the HTTP Protocol mode, XML DOM objects on a Web page are accessed through collections of `wlXmIs` objects. The `loadXML()` function is a method of the following objects:

`wlXmIs`

Description

Call `loadXML(XMLDocString)` to load XML documents into XML DOM objects. This allows users to work with XML documents and data that did not originate in HTML Data Islands, such as with Native Browsing. In a typical scenario, a user downloads an XML document. WebLOAD saves the document contents in string form. The string is then used as the parameter for `loadXML()`. The information is loaded automatically into an XML object.

Notice that creating a new, blank XML DOM object with `WLXmlDocument()` and then loading it with a parsed XML string using `loadXML()` is essentially equivalent to creating a new XML DOM object and loading it immediately using `WLXmlDocument(xmlStr)`. As with the `WLXmlDocument(xmlStr)` constructor, only standalone, self-contained DTD strings may be used for the `loadXML()` parameter. External references in the DTD section are not allowed.

Syntax

```
loadXML(XMLDocStr)
```

Parameters

`XMLDocStr`—String parameter that contains a literal XML document in string format.

Return Values

None

Example

```
//create a new XML document object
NewXMLObj = new WLXmlDocument()
wlHttp.SaveSource = "Yes"
wlHttp.Get("http://www.server.com/xm1s/doc.xml")
XMLDocStr = document.wlSource

//load the new object with XML data
//from the saved source. We are assuming
//no external references, as explained above
NewXMLObj.loadXML(XMLDocStr)
```

Comment

You may use `loadXML()` repeatedly to load and reload XML data into XML DOM objects. Remember that each new ‘load’ into an XML DOM object will overwrite any earlier data stored in that object.

WebLOAD recommends managing XML DOM objects on a Web page through the standard `document.all` collection rather than using the `wlXm1s` family of objects.

See also

Collections, page 47

id, page 143

InnerHTML, page 153

load(), page 170

loadXML(), page 175

load() and loadXML() method comparison, page 172

src, page 255

wlXm1s, page 370

XMLDocument, page 380

location (object)

Property of Objects

The `location` object is itself a property of the following objects:

document

window

Description

A `location` object stores the parsed URL and location data of the frame or root window. For an overview of parsing, see *Parsing Web pages*, page 218 in the *WebLOAD Programming Guide*.

`location` objects are local to a single thread. You cannot create new `location` objects using the JavaScript `new` operator, but you can access HTML locations through the properties and methods of the standard DOM objects. The properties of `location` are read-only.

Syntax

Access the location's properties directly using the following syntax:

```
document.location.<location-property>
```

Note that `location.href` contains the entire URL. `href` is the default property for the location object. For example, if

```
location='http://microsoft.com'
```

then the following two URL specifications are equivalent:

```
mylocation=location.href
```

and

```
mylocation=location
```

Properties

When working in HTTP Protocol mode, note that the properties of `location` are identical to those of `link`. The only exception is that `location` has no `target` property. Note that the `location` object is not part of any collection. The `location` properties are listed below for reference.

hash, page 137

hostname, page 141

id, page 143

Name, page 184

port, page 202

search, page 230

Url, page 302

host, page 141

href, page 142

InnerText, page 155

pathname, page 199

protocol, page 208

title, page 296

wlSearchPair, page 356

Comment

Note that the `href` property contains the entire URL. The other `location` properties contain portions of the URL. `location.href` is the default property for the `location` object. For example, if

```
location='http://microsoft.com'
```

then the following two URL specifications are equivalent:

```
mylocation=location.href
```

or

```
mylocation=location
```

See also

document, page 79

link, page 169

window, page 324

map (object)

Description

Each `map` object stores the coordinate data for a client-side image map. The map objects are accessed through collections of `document.all`.

Syntax

```
<NA>
```

Properties

id

Name

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Collections, page 47

FindObject(), page 102

id, page 143

length, page 167

Name, page 184

MatchBy (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*TableCompare***Description**

Specifies how the rows to be compared should be identified. Note that to use the `MatchBy` property, the `CompareRows` property must be defined as `"all"`.

WebLOAD offers the following options:

- Compare By Row Numbers
- Compare By Column

Syntax

String ["row order"] (default)

or

Array of any combination of:

- Integer column numbers [*x*, *y*, *z*, etc.]
- String range of column numbers ["1-3", "6-8", etc.]
- String column names ["ColumnA", "ColumnB", etc.]

Example

```
a.MatchBy = ["ColumnA", "ColumnC"]
```

Comment

The `MatchBy` property is used with the `TableCompare` Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

cell, page 35 (`wlTable` and `row` property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

id, page 143 (`wlTable` property)

InnerText, page 155 (`cell` property)

Prepare(), page 205

row, page 223 (`wlTable` property)

TableCompare, page 286

wlTable, page 364

cellIndex, page 37 (`cell` property)

cols, page 48 (`wlTable` property)

CompareColumns, page 51

Details, page 72

InnerHTML, page 153 (`cell` property)

MatchBy, page 178

ReportUnexpectedRows, page 216

rowIndex, page 225 (`row` property)

tagName, page 289 (`cell` property)

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

MaxLength (property)

Property of Object

element

TextArea

Description

The maximum number of characters the user can enter into a Text or Password element.

Syntax

<NA>

Example

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

element, page 83

TextArea, page 292

Message functions

Description

These functions display messages in the Log Window of the Visual AAT or Console. Some of the functions raise errors and interrupt test session execution. For information on using the Log Window and on message types, see the *WebLOAD User's Guide*.

Example

In the following example, the Agenda attempts to download an HTML page. If it fails on the first try, it pauses for 3 minutes and tries again. If it fails on the second try, it aborts the current round.

```

function InitClient() {
    wlLocals.Url = "http://www.ABCDEF.com/index.html"
}

//First try

wlHttp.Get()
if (document.wlStatusNumber != 200) {
    InfoMessage("Thread " + ClientNum.toString() +
               " pausing for 3 min")

    Sleep(180000)

//Second try

wlHttp.Get()
if (document.wlStatusNumber != 200) {
    ErrorMessage("Aborting round " + RoundNum.toString() +
                " of thread " + ClientNum.toString())
} // End of second try
}

```

GUI mode

Note that message functions are usually accessed and inserted into Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates a message icon inserted into an Agenda Tree. The message icon is selected. The Properties pane to the right displays the properties for this message, including the message text string and the severity level of the message. The JavaScript code line that corresponds to this message function appears in the JavaScript View pane in the center.

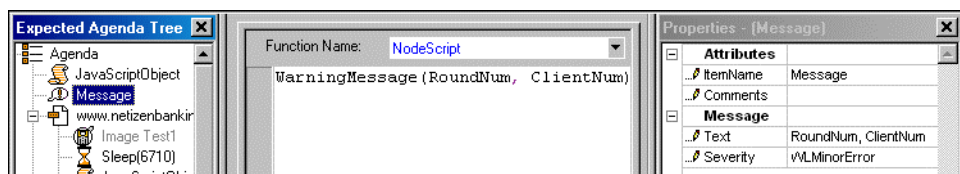


Figure 2-23: Adding a Warning Message to an Agenda

Message function command lines may also be added directly to the code in a JavaScript Object within an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Users who are programming their own JavaScript Object code within their Agenda may still take advantage of the Visual AAT GUI to simplify their programming efforts. Rather than manually type out the code for a message function, with the risk of making a mistake, even a trivial typo, and adding invalid code to the Agenda file, users may select the Edit | Insert | JavaScript | Message Commands menu to bring up a list of available message functions. The Visual AAT automatically inserts the correct code for the selected function into the JavaScript Object currently being edited. The user may then change the message text without any concerns about mistakes in the function syntax.

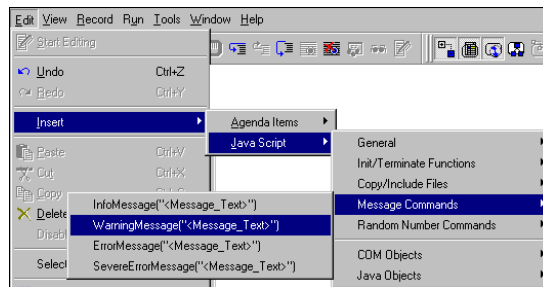


Figure 2-24: Message Commands list

See also

- | | |
|--|---|
| <i>Error Management</i> , page 104 in the <i>WebLOAD Programming Guide</i> | <i>ErrorMessage()</i> , page 89 |
| <i>ExceptionEnabled</i> , page 92 | <i>GetMessage()</i> , page 129 |
| <i>GetSeverity()</i> , page 133 | <i>InfoMessage()</i> , page 152 |
| <i>LiveConnect Overview</i> , page 255 in the <i>WebLOAD Programming Guide</i> | <i>Message functions</i> , page 180 |
| <i>ReportLog()</i> , page 215 | <i>SevereErrorMessage()</i> , page 245 |
| <i>Using the IntelliSense JavaScript Editor</i> , page 18 | <i>WarningMessage()</i> , page 323 |
| <i>wlException</i> , page 331 | <i>wlException()</i> object constructor, page 333 |

method (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

form

Description

Specifies the method that the browser should use to send the form data to the server (read-only string). A value of "Get" will append the arguments to the action URL and open it as if it were an anchor. A value of "Post" will send the data through an HTTP Post transaction. The default is "Post".

Syntax

<NA>

See also

form, page 102

MultiIPSupport (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlLocals

wlHttp

Description

WebLOAD enables use of all available IP addresses. This allows testers to simulate clients with different IP addresses using only one Load Generator.

Probing Clients use only one IP address. Load Generators are set by default to use only one IP address, but may be set to use multiple IP addresses through the keyword `MultiIPSupport`.

The possible values of `wlGlobals.MultiIPSupport` are:

- ◆ No—Use only one IP address. (default)
- ◆ Yes—Use all available IP addresses.

When connecting Load Generators through a modem, `MultiIPSupport` should be set to "No".

Example

<NA>

Comment

When the Load Generator has more than one IP address (one or more addresses on a network interface card or one or more network interface cards) WebLOAD uses ALL of the available IP addresses. Before setting `MultiIPSupport` to "Yes", make sure that all of the Applications Being Tested to which the Agenda refers are accessible through all the network interface cards.

Use the `GetIPAddress()` method to check the identity of the current IP address.

See also

Browser configuration components, page 30

GetIPAddress(), page 123

Rules of scope for local and global variables,
page 111 in the *WebLOAD Programming Guide*

wlLocals, page 343

wlGlobals, page 339

wlHttp, page 342

Name (property)

Property of Objects

Button

element

form

Image

InputCheckbox

InputImage

InputRadio

length

location

Radiobutton

text

UIContainer

wlMeta

Checkbox

File

frames

InputButton

InputFile

InputPassword

InputText

link

map

Select

TextArea

window

Description

Sets or retrieves the identification string of the parent object. Note that you can access a collection member either by its index number or by its HTML name attribute.

When working with a `wlMetas` collection, the `Name` property holds the value of the `NAME` attribute of the `META` tag.

When working with an `elements` collection, the `Name` property holds the HTML name attribute of the form element (read-only string). This is the identification string for elements of type `Button`, `CheckBox`, `File`, `Image`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text`, and `TextArea`. The `name` attribute is required. If a form element does not have a name, WebLOAD does not include it in the `elements` collection.

Syntax

Collection members may be accessed either through an index number or through a member name, if it exists. For example:

Access the first child window on a Web page using the following expression:


```
frames[0]
```

Access the first child window's link objects directly using the following syntax:

```
frames[0].frames[0].links[#].<property>
```

Alternatively, you may access a member of the `frames` collection by its HTML name attribute. For example:

```
document.frames["namestring"]
```

OR

```
document.frames.namestring
```

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

element, page 83

form, page 102

httpEquiv, page 143

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

location, page 176

Name, page 184

Select, page 231

TextArea, page 292

Url, page 302

wlMeta, page 346

Checkbox, page 38

content, page 56

File, page 99

frames, page 107

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

link, page 169

map, page 178

Radiobutton, page 210

text, page 291

UIContainer, page 301

window, page 324

Navigate() (action)

Method of Object

wlBrowser

Description

Navigate to a new URL, resetting the current active window.

Syntax

```
wlBrowser.Navigate("URL")
```

Parameters

URL—String containing the URL of the window to which we are navigating.

Return Value

None

Example

```
wlBrowser.Navigate("www.abc.com")
```

See also

Actions, page 20

AutoNavigate(), page 26

Back(), page 27

Dynamic Object Recognition (DOR) components, page 81

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

event, page 91

ExpectNavigation(), page 97

FindObject(), page 102

Forward(), page 106

Navigate(), page 186

ObjectProperty[], page 189

OnClick, page 190

OnMouseOver, page 191

Refresh(), page 213

SetWindow(), page 244

wlBrowser, page 325

wlClick(), page 326

wlClose(), page 329

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

wlMultiSelect(), page 350

wlSelect(), page 358

wlSubmit(), page 362

wlTypeIn(), page 366

NTUserName, NTPassWord (properties)

Mode

These properties are usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

wlGlobals

wlLocals

wlHttp

Description

The user name and password that the Agenda uses for Windows NT Challenge response authentication. (NT Challenge Response)

Use the `NTUserName` and `NTPassWord` properties for the Windows NT Challenge Response authentication protocol (see *DefaultAuthentication*, page 66). If the server uses the basic user authentication protocol, use the `UserName` and `PassWord` properties.

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box, illustrated in *DefaultAuthentication*, page 66.

Syntax

You may also set NT user values using the `wlGlobals` properties. For example:

```
wlGlobals.NTUserName = "Bill"
wlGlobals.NTPassWord = "Classified"
```

Comment

WebLOAD automatically sends the user name and password when a `wlHttp` object connects to an HTTP site. If an HTTP server requests NT Challenge Response authentication and you have not assigned values to `NTUserName` and `NTPassWord`, WebLOAD submits the Windows NT user name and password under which the Agenda is running.

See also

Browser configuration components, page 30

Dialog box properties, page 73

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Num() (method)

Method of Object

wlRand

Description

Return a random integer.

Syntax

```
wlRand.Num ( [seed] )
```

Parameters

[seed]—Optional seed integer used on first call to this method only if there was no previous call to the `wlRand.Seed()` method.

Return Value

A random integer.

Example

```
wlRand.Num (12345)
```

See also

Num(), page 188

Range(), page 211

Seed(), page 231

Select(), page 232

wlRand, page 355

ObjectProperty[] (property)

Property of Object

wlBrowser

Description

Array of object properties, identified by name strings. Used together with the DOR `wlBrowser.FindObject()` method to find GUI elements on the Web page, excluding form elements and map arrays. Typical object properties include `Id`, `ContainerTag`, `classid`, etc.

Syntax

```
wlBrowser.ObjectProperty["FieldName"] = "FieldValue"
```

Example

```
wlBrowser.ObjectProperty["Id"] = "MyID"
wlBrowser.ObjectProperty["ContainerTag"] = "OBJECT"
```

See also

Browser configuration components, page 30

Dialog box properties, page 73

Dynamic Object Recognition (DOR) components, page 81

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

FindObject(), page 102

Navigate(), page 186

SetWindow(), page 244

Verification Test Components, page 315

wlBrowser, page 325

Objects

Description

Chapter 1, *Introduction to JavaScript Agendas*, page 3, presents an overview of the Document Object Model (DOM), describing some of the basic objects used by standard Web browsers when working with HTML Web pages. The classic browser DOM includes a wide range of objects, properties, and methods for maximum utility and versatility. For more information about the standard DOM structure and components, go to the following Web sites:

- ◆ <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/introduction.html>
- ◆ <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dom/domoverview.asp>
- ◆ <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/dhtmlrefs.asp>

Since WebLOAD emulates the browser activities included in a test session, WebLOAD supports the standard DOM object set that implements those activities. Only the DOM objects, properties, and methods of special interest to WebLOAD programmers working with test session Agendas are listed here. This manual also includes reference material for the objects, properties, and methods that were added by WebLOAD as extensions to the basic DOM, to implement specific test session features.

Web site testing usually means testing how typical user activities are handled by the application being tested. Are the user actions managed quickly, correctly, appropriately? Is the application responsive to the user's requests? Will the typical user be happy working with this application? When verifying that an application handles user activities correctly, WebLOAD usually focuses on the user activities, recording user actions through the Visual AAT when initially creating Agendas and recreating those actions during subsequent test sessions. The focus on user activities represents a high-level, conceptual approach to test session design.

Sometimes a tester may prefer to use a low-level, "nuts-and-bolts" approach that focuses on specific internal implementation commands, such as HTTP transactions. The WebLOAD DOM

extension set includes objects, methods, properties, and functions that support this approach. Items in the *WebLOAD JavaScript Reference Manual* that are relevant to the HTTP Transaction Mode, as opposed to the more commonly used User Activity Mode, are noted as such in the entry.

See also

Actions, page 20

OnClick (property)

Property of Objects

<i>Button</i>	<i>Div</i>
<i>link</i>	<i>script</i>
<i>Span</i>	<i>TableCell</i>
<i>UIContainer</i>	

Description

Marks the click event that occurred to the parent object or the click event for which the script is written. For example, if the user clicks with the left mouse button on a *Button*, the event for the *Button* object is set to *OnClick*.

Example

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Actions</i> , page 20	<i>AutoNavigate()</i> , page 26
<i>Back()</i> , page 27	<i>Button</i> , page 34
<i>Checkbox</i> , page 38	<i>Collections</i> , page 47
<i>event</i> , page 91	<i>File</i> , page 99
<i>form</i> , page 102	<i>Forward()</i> , page 106
<i>Image</i> , page 148	<i>InputButton</i> , page 157
<i>InputCheckbox</i> , page 158	<i>InputFile</i> , page 159
<i>InputImage</i> , page 160	<i>InputRadio</i> , page 162

<i>InputText</i> , page 164	<i>length</i> , page 167
<i>Navigate()</i> , page 186	<i>OnClick</i> , page 190
<i>OnMouseOver</i> , page 191	<i>Radiobutton</i> , page 210
<i>Refresh()</i> , page 213	<i>Select</i> , page 231
<i>SetWindow()</i> , page 244	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301
<i>wlBrowser</i> , page 325	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlSubmit()</i> , page 362	<i>wlTypeIn()</i> , page 366

OnMouseOver (property)

Property of Objects

<i>Div</i>	<i>link</i>
<i>Span</i>	<i>TableCell</i>
<i>UIContainer</i>	

Description

Marks the mouseover event that occurred to the parent object or the mouseover event for which the script is written. For example, if the user moves the mouse pointer into an `Image`, the `event` for the `Image` object is set to `OnMouseOver`.

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>Actions</i> , page 20	<i>AutoNavigate()</i> , page 26
<i>Back()</i> , page 27	<i>Button</i> , page 34
<i>Checkbox</i> , page 38	<i>Collections</i> , page 47
<i>event</i> , page 91	<i>File</i> , page 99
<i>form</i> , page 102	<i>Forward()</i> , page 106
<i>Image</i> , page 148	<i>InputButton</i> , page 157

<i>InputCheckbox</i> , page 158	<i>InputFile</i> , page 159
<i>InputImage</i> , page 160	<i>InputRadio</i> , page 162
<i>InputText</i> , page 164	<i>length</i> , page 167
<i>Navigate()</i> , page 186	<i>OnClick</i> , page 190
<i>OnMouseOver</i> , page 191	<i>Radiobutton</i> , page 210
<i>Refresh()</i> , page 213	<i>Select</i> , page 231
<i>SetWindow()</i> , page 244	<i>text</i> , page 291
<i>TextArea</i> , page 292	<i>UIContainer</i> , page 301
<i>wlBrowser</i> , page 325	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlSubmit()</i> , page 362	<i>wlTypeIn()</i> , page 366

Open() (function)

Method of Object

wlOutputFile

Description

Opens a file. When called as a stand-alone function, opens an input file. When called as a method of the *wlOutputFile* object, opens an output file.

Syntax

Stand-Alone function:

```
Open(filename, FileReadFlag)
```

For example:

```
InitAgenda() {
    Open("MyInitFile.txt", WLRandom)
}
```

wlOutputFile method:

```
MyFileObj = new wlOutputFile(filename)
...
MyFileObj.Open()
```


Parameters**Stand-Alone function:**

filename—A string with the name of the ASCII input file to be opened.

FileReadFlag—One of two flags signifying whether the input file should be read sequentially (*WLSequential*, default) or randomly (*WLRandom*).

wlOutputFile method:

None

Return Value

None.

Comment

The `Open()` function must be included in the `InitAgenda()` function with the `WLRandom` flag set if you wish random access to the input file. If you prefer the standard sequential access then the function is not required.

See also

Close(), page 45

delete(), page 70

GetLine(), page 124

Open(), page 193

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

wlOutputFile, page 351

Write(), page 372

CopyFile(), page 57

File management functions, page 100

IncludeFile(), page 150

Reset(), page 219

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile(), page 354

Writeln(), page 373

OpenStream() (method)

Method of Object

wlMediaPlayer

Description

This method opens a stream file for reading and playback.

Syntax

```
MyMediaPlayerObject.OpenStream(streamName)
```

Parameters

`streamName`—String containing the name of the stream to access.

Return Value

None.

Example

<NA>

Comment

If `streamName` is empty (“”), the `fileName` property must be set before calling this method.

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

option (object)

Property of Object

`Option` objects are grouped into collections of `options` that are themselves properties of the following:

element

Select

Description

A collection of the nested <OPTION> objects only found within elements of type SELECT, i.e., `forms[n].elements[n].type = "SELECT"`. Each `option` object denotes one choice in a `select` element, containing information about a selected form element.

`option` objects are local to a single thread. You cannot create new `option` objects using the JavaScript `new` operator, but you can access HTML options through the properties and methods of the standard DOM objects. `option` properties are read-only.

option objects are grouped together within collections of options. To access an individual option's properties, check the length property of the options collection and use an index number to access the individual options.

Syntax

To find out how many option objects are contained within a form element, check the value of:

```
document.forms[#].elements[#].options.length
```

Access each option's properties directly using the following syntax:

```
document.forms[#].elements[#].options[#].<option-property>
```

For example:

```
document.forms[1].elements[2].options[0].selected
```

Comment

Note that options only exist if the type of the parent element is <SELECT>, i.e., forms[n].elements[n].type = "SELECT". For example, to check whether a form element is of type <SELECT> and includes an options collection, you could use the following Agenda:

```
function InitAgenda()
{
    wlGlobals.Proxy = "webproxy.xyz.com:8080"
    // Through proxy

    wlGlobals.SaveSource = "Yes"
    wlGlobals.ParseForms = "Yes"
    wlGlobals.ParseTables = "Yes"
}

function CheckElementType(WebTestSite)
{
    wlHttp.Get(WebTestSite)
    if (document.forms.length > 0)
        if (document.forms[0].elements.length > 0)
        {
            InfoMessage("We have a candidate. " +
                "Element type is " +
                document.forms[0].elements[0].type)

            InfoMessage
            ("document.forms[0].elements[0].options.length is "
                + document.forms[0].elements[0].options.length)
        }
    }

    CheckElementType("http://www.TestSite1.com/domain/pulldown.htm")
}
```

```
CheckElementType ("http://www.TestSite2.com/")
ErrorMessage ("Done!")
```

Properties*defaultselected*, page 68*selected*, page 233*text*, page 291*value*, page 310**See also***element*, page 83*Select*, page 231

OuterLink (property)

Property of Objects*Image***Description**Represents the outer link field for the parent *image* object.**Syntax**

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also*Button*, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99*form*, page 102*Image*, page 148*InputButton*, page 157*InputCheckbox*, page 158*InputFile*, page 159*InputImage*, page 160*InputRadio*, page 162*InputText*, page 164*length*, page 167*Radiobutton*, page 210*Select*, page 231*text*, page 291*TextArea*, page 292*UIContainer*, page 301

Outfile (property)

Property of Objects

wlBrowser

Description

The name of a file to which WebLOAD writes response data from the HTTP server.

The `Outfile` will contain the data from the *next* HTTP transaction, so the `Outfile` command must *precede* the next transaction.

The default is "", which means do not write the response data.

If there is more than one transaction after the `Outfile` property, only the response data from the *first* transaction will be written. To write the response data from each transaction an `Outfile` statement must be placed PRIOR to *each* transaction.

The `Outfile` property is independent of the `SaveSource` property. `Outfile` saves in a file. `SaveSource` stores the downloaded data in `document.wlSource`, in memory.

Example

To write the response data from
"http://note/radview/radview.html" in
"c:\temp.html"
you might write:

```
wlHttp.Outfile = "c:\\temp.html"  
wlHttp.Get("http://note/radview/radview.html")
```

Comment

The `Outfile` property saves *server response data*. To save *Agenda output messages*, use the `wlOutputFile`.

See also

wlOutputFile, page 351

wlBrowser, page 325

PassWord (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects*wlGlobals**wlLocals**wlHttp***Description**

The password that the Agenda uses to log onto a restricted HTTP site. WebLOAD automatically uses the appropriate access protocol. For example, if a site expects clients to use the NT Authentication protocol, the appropriate user name and password will be stored and sent accordingly.

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box, illustrated in *DefaultAuthentication*, page 66.

Syntax

You may also set user values using the *wlGlobals* properties. WebLOAD automatically sends the user name and password when a *wlHttp* object connects to an HTTP site. For example:

```
wlGlobals.UserName = "Bill"
wlGlobals.Password = "TopSecret"
```

See also

Browser configuration components, page 30

Dialog box properties, page 73

Rules of scope for local and global variables,
page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

pathname (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

link

location

Description

The URI portion of the URL, including the directory path and filename (read-only string).

Example

```
"/products/order.html"  
"/search.exe"
```

See also

link, page 169

location, page 176

Pause() (method)

Method of Object

wlMediaPlayer

Description

This method temporarily pauses the current play operation without changing the logical stream position.

Syntax

```
MyMediaPlayerObject.Pause()
```

Parameters

None

Return Value

None.

Example

<NA>

See also*bitrate*, page 30*currentPosition*, page 62*duration*, page 80*OpenStream()*, page 194*Play()*, page 200*state*, page 277*type*, page 300*connectionBandwidth*, page 53*currentStreamName*, page 62*fileName*, page 100*Pause()*, page 200*Resume()*, page 220*Stop()*, page 278*wlMediaPlayer*, page 344

Play() (method)

Method of Object*wlMediaPlayer***Description**

This method starts playback at the point of the specified starting time offset.

Syntax

```
MyMediaPlayerObject.Play(startPosition, duration)
```

Parameters

startPosition—Specifies the starting position within the stream in milliseconds. Player will seek to the correct position in the stream. If *startPosition* is set to `CURRENTPOSITION`, playback will start from the current position in the stream. If no position is specified, playback will start from the beginning of the stream.

duration—Specifies the total stream length in milliseconds. In addition to passing an explicit number, one of two global flag options may be used:

- `ASYNCH_PLAY`—function will return immediately. Test session continues execution. The stream continues playing in the background until the end of the current round or until the `Stop()` method is called, whichever comes first. (default)
- `INFINITE_PLAY`—function will play to the end of the file and not return. Test session is held (paused) until the end of the stream is reached. If the stream is live and does not end, the test session will be held indefinitely.

Player will seek to the correct position in the stream. If *startPosition* is set to `CURRENTPOSITION`, playback will start from the current position in the stream. If no position is specified, playback will start from the beginning of the stream.

Return Value

None.

Example

<NA>

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

port (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

link

location

Description

The port of the URL (read-only integer).

Example

80

See also

link, page 169

location, page 176

Post() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

wlHttp

Description

Perform an HTTP or HTTPS Post command. The method sends the `FormData`, `Data`, or `DataFile` properties in the Post command. In this way, you can submit any type of data to an HTTP server.

Syntax

```
Post([URL] [, TransName])
```

Parameters

[URL]—An optional parameter identifying the document URL.

You may optionally specify the URL as a parameter of the method. Post() connects to first URL that has been specified from the following list:

1. A `Url` parameter specified in the method call.
2. The `Url` property of the `wlHttp` object.
3. The local default `wlLocals.Url`.
4. The global default `wlGlobals.Url`.

The URL must be a server that accepts the posted data.

[TransName]—An optional user-supplied string with the transaction name as it will appear in the Statistics Report, described in *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*.

Use *named transactions* to identify specific HTTP transactions by name. This simplifies assigning counters when you want WebLOAD to automatically calculate a specific transaction's occurrence, success, and failure rates.

The run-time statistics for transactions to which you have assigned a name appear in the Statistics Report. For your convenience, WebLOAD offers an Automatic Transaction option. On the Console, select Automatic Transaction from the General Tab of the Global Options dialog box. Automatic Transaction is set to True by default. With Automatic Transaction, WebLOAD automatically assigns a name to every Get and Post HTTP transaction. This makes statistical analysis simpler, since all HTTP transaction activity is measured, recorded, and reported for you automatically. You do not have to remember to

add naming instructions to each Get and Post command in your Agenda. The name assigned by WebLOAD is simply the URL used by that Get or Post transaction. If your Agenda includes multiple transactions to the same URL, the information will be collected cumulatively for those transactions.

Return Value

None

Example

```
function InitAgenda()
    //Set the default URL
    wlGlobals.Url = "http://www.ABCDEF.com"
}
//Main script
//Connect to the default URL:
wlHttp.Post()
//Connect to a different, explicitly set URL:
wlHttp.Post("http://www.ABCDEF.com/product_info.html")
//Assign a name to the following HTTP transact:
wlHttp.Get("http://www.ABCDEF.com/product_info.html",
    "UpdateBankAccount")
//Submit to a CGI program
wlHttp.Url = "http://www.ABCDEF.com/search.cgi"
wlHttp.FormData["SeachTerm"] = "ocean+currents"
wlHttp.Post()
//Submit to an HTTP server of any type
wlHttp.FormData["FirstName"] = "Bill"
wlHttp.FormData["LastName"] = "Smith"
wlHttp.Post("http://www.ABCDEF.com/formprocessor.exe")
```

Use named transactions as a shortcut in place of the `BeginTransaction()...EndTransaction()` module. For example, this is one way to identify a logical transaction unit:

```
BeginTransaction("UpdateBankAccount")
    wlHttp.Get(url)
    ... // the body of the transaction
    ... // any valid JavaScript statements
    wlHttp.Post(url);
EndTransaction("UpdateBankAccount")
... // and so on
```

Using the named transaction syntax, you could write:

```
wlHttp.Get(url, "UpdateBankAccount")
... // the body of the transaction
... // any valid JavaScript statements
```

```
wlHttp.Post(url, "UpdateBankAccount")
... // and so on
```

For the HTTPS protocol, include "https://" in the URL and set the required properties of the `wlGlobals` object:

```
wlHttp.Post("https://www.ABCDEF.com")
```

The URL can contain a string of attribute data.

```
wlHttp.Post("http://www.ABCDEF.com/query.exe"+
"?SearchFor=icebergs&SearchType=ExactTerm")
```

Alternatively, you can specify the attributes in the `FormData` or `Data` property. The method automatically appends these in the correct syntax to the URL. Thus the following two code fragments are each equivalent to the preceding `Post` command.

```
wlHttp.Data.Type = "application/x-www-form-urlencoded"
wlHttp.Data.Value = "SearchFor=icebergs&SearchType=ExactTerm"
wlHttp.Post("http://www.ABCDEF.com/query.exe")
```

or

```
wlHttp.FormData.SearchFor = "icebergs"
wlHttp.FormData.SearchType = "ExactTerm"
wlHttp.Post("http://www.ABCDEF.com/query.exe") <NA>
```

Comment

You may not use the `TransName` parameter by itself. `Post()` expects to receive either *no* parameters, in which case it uses the Agenda's default URL, or *one* parameter, which must be an alternate URL value, or *two* parameters, including both a URL value and the transaction name to be assigned to this transaction.

See also

<i>Adding transactions</i> , page 49 in the <i>WebLOAD Programming Guide</i>	<i>BeginTransaction()</i> , page 28
<i>CreateDOM()</i> , page 59	<i>CreateTable()</i> , page 60
<i>Data</i> , page 63	<i>Data Drilling—WebLOAD transaction reports</i> , page 149 in the <i>WebLOAD Programming Guide</i>
<i>DataFile</i> , page 65	<i>FormData</i> , page 104
<i>Functional Testing and Reporting</i> , page 127 in the <i>WebLOAD Programming Guide</i>	<i>ReportEvent()</i> , page 214
<i>SetFailureReason()</i> , page 239	<i>VerificationFunction()</i> (user-defined), page 320
<i>wlHttp</i> , page 342	

Prepare() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

TableCompare

Description

This method prepares the expected table associated with a `TableCompare` object for later comparison using the `Compare` method. Used in the `InitClient()` function, right after creating a new `TableCompare` object.

Syntax

```
Prepare ()
```

Parameters

None

Return Value

None

Example

<NA>

Comment

The `Prepare()` method is used with the `TableCompare` Wizard, which is only available to users of the manual AAT Agenda recording program.. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

<i>cell</i> , page 35 (<code>wlTable</code> and <code>row</code> property)	<i>cellIndex</i> , page 37 (<code>cell</code> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<code>wlTable</code> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<code>wlTable</code> property)	<i>InnerHTML</i> , page 153 (<code>cell</code> property)
<i>InnerText</i> , page 155 (<code>cell</code> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216

row, page 223 (*wlTable* property)

TableCompare, page 286

wlTable, page 364

rowIndex, page 225 (*row* property)

tagName, page 289 (*cell* property)

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

ProbingClientThreads (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

wlGlobals

wlLocals

wlHttp

Description

Optionally, WebLOAD can allocate extra threads to download nested images and frames.

For clients that you define in a Load Generator, this option is controlled by the *LoadGeneratorThreads* property. The default value of this property is "Single", which means that Virtual Clients will not use extra threads to download data from the Server.

For the Probing Client, the option is controlled by the *ProbingClientThreads* property. The default is "Multiple", which means that the client can use three extra threads for nested downloads. This simulates the behavior of Web browsers, which often use extra threads to download nested images and frames.

The possible values of these properties are:

- ◆ *Single*—Do not use extra threads to download nested images and frames. (default for *LoadGeneratorThreads*)
- ◆ *Multiple*—Allocate three extra threads per client (for a total of four threads per client) to download nested images and frames. (default for *ProbingClientThreads*)
- ◆ Any specific number of threads between 1 and 8, such as "5"—Allocate that exact number of extra threads per client to download nested images and frames.

Example

You can assign any of these properties independently within a single Agenda. In that case, if you configure a Probing Client to run the Agenda, WebLOAD uses the value of *ProbingClientThreads* and ignores *LoadGeneratorThreads* (vice versa if you configure a Load Generator to run the Agenda). For example, you might write:

```
function InitAgenda() {
    //Do not use extra threads if a
    // Probing Client runs the Agenda
    wlGlobals.ProbingClientThreads = "Single"

    //Use extra threads if a
    // Load Generator runs the Agenda
    wlGlobals.LoadGeneratorThreads = "Multiple"
}
```

Comment

The extra threads have no effect on the `ClientNum` value of the client. The `ClientNum` variable reports only the main thread number of each client, not the extra threads.

GUI mode

WebLOAD recommends enabling or disabling multi-threaded virtual clients through the Console GUI. Enable multi-threading for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box and setting the number of threads you prefer, as illustrated in the following figure:

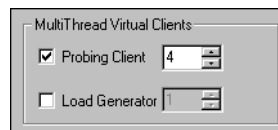


Figure 2-25: Enabling multi-threading for Load Generator

See also

Browser configuration components, page 30

LoadGeneratorThreads, page 173

Rules of scope for local and global variables,
page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

protocol (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

Image

link

location

Description

The HTTP protocol portion of the URL for the parent object (read-only string).

Example

```
"https://"
```

See also

Image, page 148

link, page 169

location, page 176

Proxy, ProxyUserName, ProxyPassWord (properties)

Mode

The following properties are usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

wlGlobals

wlLocals

wlHttp

Description

Identifies the proxy server that the Agenda uses for HTTP access. The user name and password are for proxy servers that require user authorization.

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box, illustrated in *DefaultAuthentication*, page 66.

Syntax

You may also set proxy user values using the `wlGlobals` properties. WebLOAD automatically connects via the proxy when a `wlHttp` object connects to an HTTP site.

```
wlGlobals.ProxyProperty = "TextString"
```

Example

```
wlGlobals.Proxy = "proxy.ABCDEF.com:8080"
wlGlobals.ProxyUserName = "Bill"
wlGlobals.ProxyPassword = "Classified"
```

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlLocals, page 343

Dialog box properties, page 73

Security, page 89 in the *WebLOAD Programming Guide*

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Radiobutton (object)

Property of Objects

`Radiobutton` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Each `Radiobutton` object represents one of the radiobuttons embedded in a document. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["RADIO"]
```

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

AdjacentText, page 23

id, page 143

Name, page 184*value*, page 310**Comment**

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also*Button*, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99*form*, page 102*Image*, page 148*InputButton*, page 157*InputCheckbox*, page 158*InputFile*, page 159*InputImage*, page 160*InputRadio*, page 162*InputText*, page 164*length*, page 167*Radiobutton*, page 210*Select*, page 231*text*, page 291*TextArea*, page 292*UIContainer*, page 301

Range() (method)

Method of Object*wlRand***Description**

Return a random integer between *start* and *end*.

Syntax

```
wlRand.Range(start, end, [seed])
```

Parameters

start—Integer signifying start of specified range of numbers.

end—Integer signifying end of specified range of numbers.

[*seed*]—Optional seed integer used on first call to this method only if there was no previous call to the *wlRand.Seed()* method.

Return Value

A random integer that falls within the specified range.

Example

```
wlRand.Num(12345)
```

See also

Num(), page 188

Range(), page 211

Seed(), page 231

Select(), page 232

wlRand, page 355

ReceiveTimeout (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlLocals

wlHttp

Description

If the Agenda is not able to connect to a server, the Agenda will wait and then retry to establish a connection. *ReceiveTimeout* is the amount of time, in milliseconds, that the Agenda will wait before attempting to reconnect.

GUI mode

Connection timeout values should be set through the Visual AAT or Console GUI.

Example

```
wlBrowser.ReceiveTimeout = 900000
```

See also

Browser configuration components, page 30

Dialog box properties, page 73

RequestRetries, page 217

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

RedirectionLimit (property)

Property of Objects

wlBrowser

Description

The maximum number of redirection ‘hops’ allowed during a test session. The default value is 10.

GUI mode

WebLOAD recommends setting the redirection limit through the Console GUI. Check Redirection Enabled and enter a limiting number on the Browser Emulation tab of the Tools | Default or Current Options dialog box, as illustrated in the following figure:

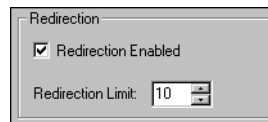


Figure 2-26: Setting Redirection Limit

Syntax

You may also assign a redirection limit value using the `wlBrowser.RedirectionLimit` property.

```
wlBrowser.RedirectionLimit = IntegerValue
```

Example

```
wlBrowser.RedirectionLimit = 10
```

See also

wlBrowser, page 325

Refresh() (action)

Method of Objects

wlBrowser

Description

Emulate clicking on the refresh button in the browser to reload the current page.

Syntax

```
wlBrowser.Refresh()
```

Parameters

None

Return Value

None

Example

<NA>

Comment

This method implements a special category of user action—clicking on a specific browser shortcut button from the Microsoft IE toolbar.

See also

Actions, page 20

Back(), page 27

Forward(), page 106

OnClick, page 190

Refresh(), page 213

wlBrowser, page 325

wlClose(), page 329

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlSubmit(), page 362

AutoNavigate(), page 26

event, page 91

Navigate(), page 186

OnMouseOver, page 191

SetWindow(), page 244

wlClick(), page 326

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

wlTypeIn(), page 366

ReportEvent() (function)

Description

This function enables you to record specific events as they occur. This information is very helpful when analyzing Web site performance with Data Drilling. (See *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*, for more information.)

Syntax

```
ReportEvent (EventName [, description])
```

Parameters

EventName—A user-supplied string that identifies the specific event and appears in the results tables of the WebLOAD Data Drilling feature. Since this name is used as a table header and sort key, it must be a short string that is used consistently to identify events, such as “URLMismatch”.

[**description**]—An optional user-supplied string that may be longer and more detailed than the **EventName**, providing more information about the specific event.

Return Value

None.

Example

<NA>

See also

<i>Adding transactions</i> , page 49 in the <i>WebLOAD Programming Guide</i>	<i>BeginTransaction()</i> , page 28
<i>CreateDOM()</i> , page 59	<i>CreateTable()</i> , page 60
<i>Custom verification functions</i> , page 137 in the <i>WebLOAD Programming Guide</i>	<i>Data Drilling—WebLOAD transaction reports</i> , page 149 in the <i>WebLOAD Programming Guide</i>
<i>EndTransaction()</i> , page 85	<i>Functional Testing and Reporting</i> , page 127 in the <i>WebLOAD Programming Guide</i>
<i>ReportEvent()</i> , page 214	<i>SetFailureReason()</i> , page 239
<i>TimeoutSeverity</i> , page 293	<i>Transaction verification components</i> , page 297
<i>TransactionTime</i> , page 298	<i>Validate()</i> , page 309
<i>Verification Test Components</i> , page 315	<i>Verification Test Property List: Global and Page Level</i> , page 318
<i>VerificationFunction()</i> (user-defined), page 320	<i>wlBrowser</i> , page 325
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

ReportLog() (method)

Method of Object

wlException

Description

Sends a message to the Log Window that includes the error message and severity level stored in this *wlException* object.

Syntax

```
ReportLog ()
```

Parameters

None

Return Value

None

Example

```
myUserException.ReportLog ()
```

See also

Error Management, page 104 in the *WebLOAD Programming Guide*

ErrorMessage(), page 89

ExceptionEnabled, page 92

GetMessage(), page 129

GetSeverity(), page 133

InfoMessage(), page 152

LiveConnect Overview, page 255 in the *WebLOAD Programming Guide*

Message functions, page 180

ReportLog(), page 215

SevereErrorMessage(), page 245

Using the IntelliSense JavaScript Editor, page 18

WarningMessage(), page 323

wlException, page 331

wlException() object constructor, page 333

ReportUnexpectedRows (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

TableCompare

Description

Specifies whether table comparison should simply be completed for rows found in the first table, compared against rows found in the second table, ignoring any extra, additional rows found in the second table (false), or whether the user should be notified about any unexpected rows found in the second table (true).

Syntax

Boolean (true/false, default true)

Example

```
a.ReportUnexpectedRows = false
```

Comment

The `ReportUnexpectedRows` property is used with the TableCompare Wizard, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

<i>cell</i> , page 35 (<code>wlTable</code> and <code>row</code> property)	<i>cellIndex</i> , page 37 (<code>cell</code> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<code>wlTable</code> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<code>wlTable</code> property)	<i>InnerHTML</i> , page 153 (<code>cell</code> property)
<i>InnerText</i> , page 155 (<code>cell</code> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<code>wlTable</code> property)	<i>rowIndex</i> , page 225 (<code>row</code> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<code>cell</code> property)
<i>wlTable</i> , page 364	
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

RequestRetries (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

<i>wlGlobals</i>	<i>wlLocals</i>
<i>wlHttp</i>	

Description

If a Virtual Client is not able to connect to a server, the Virtual Client will wait and then retry to establish a connection. `RequestRetries` is the maximum number of times that the Virtual Client will attempt to reconnect.

GUI mode

Connection retry values should be set through the Visual AAT or Console GUI.

Example

```
wlGlobals.RequestRetries = 7
```

See also

Browser configuration components, page 30

ReceiveTimeout, page 212

wlGlobals, page 339

wlLocals, page 343

Dialog box properties, page 73

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Reset

Reset (object)

Property of Objects

`Reset` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = reset`. Each `Reset` object represents a button that, when clicked, resets the form's controls to their initial values. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["RESET"]
```

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlMouseUp(), page 349

wlSelect(), page 358

Properties

event, page 91

InnerText, page 155

OnClick, page 190

value, page 310

id, page 143

Name, page 184

title, page 296

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

Reset() (method)

Method of Object

wlOutputFile()

Description

Return to the beginning of the output file.

Syntax

Reset ()

Parameters

None

Return Value

None

Example

```
MyFileObj = new wLOutputFile (filename)
...
MyFileObj.Reset ()
```

See also*Close()*, page 45*delete()*, page 70*GetLine()*, page 124*Open()*, page 193*Using the Form Data Wizard*, page 35 in the *WebLOAD Programming Guide**wLOutputFile*, page 351*Write()*, page 372*CopyFile()*, page 57*File management functions*, page 100*IncludeFile()*, page 150*Reset()*, page 219*Using the IntelliSense JavaScript Editor*, page 18*wLOutputFile()*, page 354*WriteLn()*, page 373

Resume() (method)

Method of Object*wLMediaPlayer***Description**

This method resumes playback from the current point in the stream following a `Pause()` method call.

Syntax

```
MyMediaPlayerObject.Resume ()
```

Parameters

None

Return Value

None.

Example

<NA>

See also*bitrate*, page 30*currentPosition*, page 62*connectionBandwidth*, page 53*currentStreamName*, page 62

<i>duration</i> , page 80	<i>fileName</i> , page 100
<i>OpenStream()</i> , page 194	<i>Pause()</i> , page 200
<i>Play()</i> , page 200	<i>Resume()</i> , page 220
<i>state</i> , page 277	<i>Stop()</i> , page 278
<i>type</i> , page 300	<i>wlMediaPlayer</i> , page 344

RoundNum (variable)

Description

The number of times that WebLOAD has executed the main script of a client during the WebLOAD test, including the current execution. RoundNum is a read-only local variable, reporting the number of rounds for the specific WebLOAD client, no matter how many other clients may be running the same Agenda.

RoundNum does not exist in the global context of an Agenda (`InitAgenda()`, etc.). In the local context:

- ◆ In `InitClient()`, RoundNum = 0.
- ◆ In the main script, RoundNum = 1, 2, 3,
- ◆ In `TerminateClient()`, `OnScriptAbort()`, or `OnErrorTerminateClient()`, RoundNum keeps its value from the final round.

The WebLOAD clients do not necessarily remain in synchronization. The RoundNum may differ for different clients running the same Agenda.

If a thread stops and restarts for any reason, the RoundNum continues from its value before the interruption. This can occur, for example, after you issue a Pause command from the WebLOAD Console.

If you mix Agendas in a single Load Generator, WebLOAD maintains an independent round counter for each Agenda. For example, if agenda1 has executed twice and agenda2 has executed three times on a particular thread, the RoundNum of agenda1 is 2 and the RoundNum of agenda2 is 3.

Example

<NA>

GUI mode

WebLOAD recommends accessing global system variables, including the RoundNum identification variable, through the Visual AAT GUI. To see a list of global system variables, highlight the top-level Agenda node in the Agenda Tree, find the `GlobalVariables` section of the Properties pane, and click on the browse button of the System field. A complete list of system variables and macros pops up, as illustrated in the following figure. The variables that

appear in this list are available for use at any point in an Agenda file. For example, it is convenient to add `RoundNum` to a Message Node to clarify the round in which the messages that appear in the Console Log window originated.

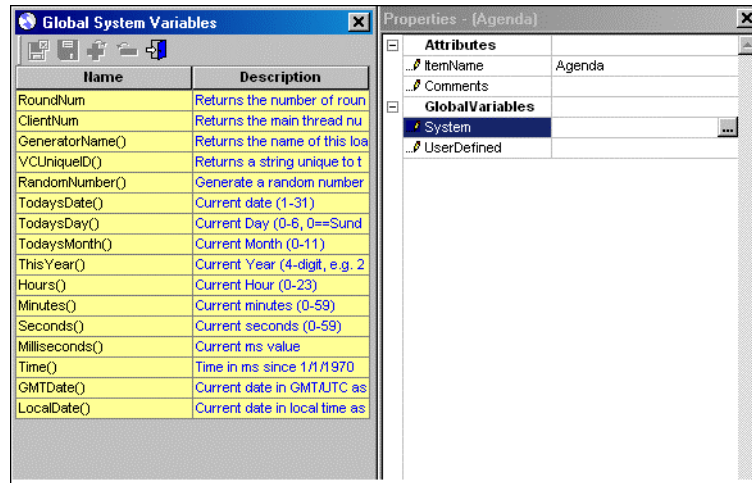


Figure 2-27: Global System Variables List

Note that `RoundNum` can also be added directly to the code in an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Select a function from the Edit | Insert | JavaScript | Copy/Include Files menu. The Visual AAT automatically inserts the correct code for the selected variable into the Agenda file. The user may then edit variable values without any concerns about mistakes in the function syntax.

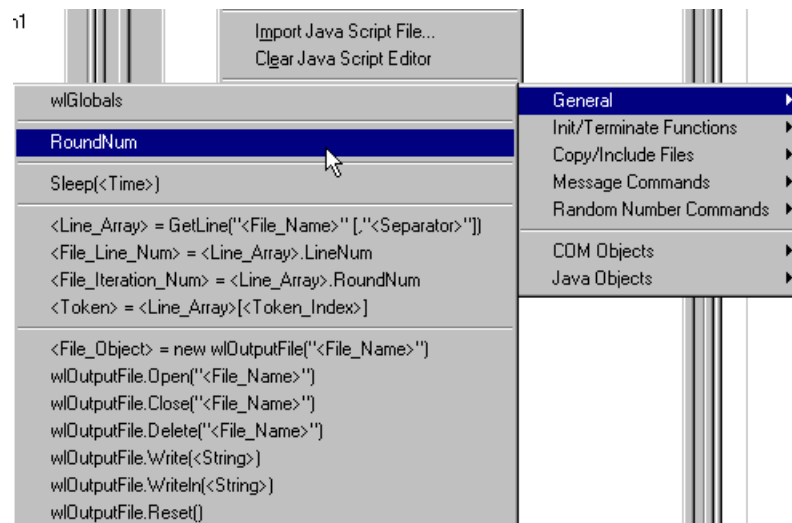


Figure 2-28: Inserting RoundNum

See also

ClientNum, page 43

GeneratorName(), page 108

GetOperatingSystem(), page 130

Identification variables and functions, page 146

RoundNum, page 221

User-defined global properties, page 304

Using the IntelliSense JavaScript Editor, page 18

VCUniqueID(), page 312

row (object)

Mode

This object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

`row` objects are grouped into collections of `rows`. The `rows` collection is a property of the following objects:

TextArea

wlTable

Description

When working with `TextArea` element objects, a `row` object contains the number of rows in the `TextArea`.

When working with `wlTables` objects, a `row` object contains all the data found in a single table row. Individual `row` objects may be addressed by index number, similar to any object within a collection.

Syntax

Elements object:

<NA>

wlTables object:

Individual `row` objects are addressed by index number, similar to any object within a collection. Access each row's properties directly using the following syntax:

```
document.wlTables.myTable.rows[#].<row-property>
```

Example

wlTables object:

To find out how many row objects are contained within `myTable`, check the value of:

```
document.wlTables.myTable.rows.length
```

To access a property of the 16th row in `myTable`, with the first row indexed at 0, you could write:

```
document.wlTables.myTable.rows[15].rowIndex
```

To access a property of the 4th cell in the 3rd row in `myTable`, counting across rows and with the first cell indexed at 0, you could write:

```
document.wlTables.myTable.rows[2].cells[3].<cell-property>
```

Properties

wlTables object:

Each `row` object contains information about the data found in the cells of a single table row. The `row` object includes the following properties:

rowIndex, page 225 (row property)

cell, page 35 (row property)

Comment

The `row` object may be accessed as a member of the `wlTables` family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

<i>cell</i> , page 35 (<i>wlTable</i> and <i>row</i> property)	<i>cellIndex</i> , page 37 (<i>cell</i> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<i>wlTable</i> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<i>wlTable</i> property)	<i>InnerHTML</i> , page 153 (<i>cell</i> property)
<i>InnerText</i> , page 155 (<i>cell</i> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<i>wlTable</i> property)	<i>rowIndex</i> , page 225 (<i>row</i> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<i>cell</i> property)
<i>wlTable</i> , page 364	<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>

rowIndex (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

row

Description

An integer containing the ordinal index number of this *row* object within the parent table. Rows are indexed starting from zero, so the *rowIndex* of the first row in a table is 0.

Syntax

<NA>

Example

<NA>

Comment

The *rowIndex* property is a member of the *wlTables* family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard

`document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

cell, page 35 (*wlTable* and *row* property)

Collections, page 47

Compare(), page 49

CompareRows, page 52

id, page 143 (*wlTable* property)

InnerText, page 155 (*cell* property)

Prepare(), page 205

row, page 223 (*wlTable* property)

TableCompare, page 286

wlTable, page 364

cellIndex, page 37 (*cell* property)

cols, page 48 (*wlTable* property)

CompareColumns, page 51

Details, page 72

InnerHTML, page 153 (*cell* property)

MatchBy, page 178

ReportUnexpectedRows, page 216

rowIndex, page 225 (*row* property)

tagName, page 289 (*cell* property)

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

SaveSource (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlLocals

wlHttp

Description

Instruct WebLOAD to store the complete HTML source code downloaded in an HTTP command.

- ◆ No—Do not store the source HTML. (default)
- ◆ Yes—Store the source HTML in `document.wlSource`.

If you enable `SaveSource`, WebLOAD automatically stores the downloaded HTML whenever the Agenda calls the `wlHttp.Get()` or `wlHttp.Post()` method. WebLOAD stores the most recent download in the `document.wlSource` property, refreshing it when the Agenda calls `wlHttp.Get()` or `wlHttp.Post()` again. The stored code includes any scripts or other data embedded in the HTML. Your Agenda can retrieve the code from `document.wlSource` and interpret it in any desired way.

Example

<NA>

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

SaveTransaction (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

Description

Instruct WebLOAD to save detailed information about all transactions, both successes and failures, for later analysis in the Data Drilling reports. (See *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*, for more information about the Data Drilling report set.)

By default, WebLOAD only saves detailed information about transaction failures for later analysis, since most test sessions are focused on tracking down and identifying the causes of errors and failures. WebLOAD also provides the option of storing and analyzing the data for all transactions in a test session, successes and failures, through the `SaveTransaction` property. However, this property should be used carefully, since a successful test session may run for an extended period, and saving data on each transaction success could quickly use up all available space. Possible values are:

- ◆ No—Do not store detailed data on successful transactions. (default)
- ◆ Yes—Store detailed data on successful transactions.

Example

```
function InitAgenda() {
    wlGlobals.SaveTransaction = True
}
```

Comment

As with all `wlGlobals` configuration properties, the `SaveTransaction` property must be set in the `InitAgenda()` function, as illustrated in the preceding example.

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

Browser configuration components, page 30

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SaveWSTransaction, page 228

wlGlobals, page 339

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

SaveWSTransaction (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

Description

Instruct WebLOAD to save detailed information about all WebServices transactions for later analysis in the Data Drilling reports, both failures and successes. (See *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*, for more information about the Data Drilling report set.) Web Services transactions are identified in the Data Drilling reports by the name of the Web Service combined with the name of the specific method being called.

By default, WebLOAD only saves detailed information about transaction failures for later analysis, since most test sessions are focused on tracking down and identifying the causes of errors and failures. WebLOAD also provides the option of storing and analyzing the data for all transactions in a test session, successes and failures, through the `SaveTransaction` property. However, this property should be used carefully, since a successful test session may

run for an extended period, and saving data on each transaction success could quickly use up all available space. For test sessions that are focusing on Web Services, WebLOAD provides the compromise option of saving data on all transaction failures, as usual, plus data on all Web Services transactions, both failures and successes. This provides maximum information about the Web Services activities while minimizing the storage space required. Possible values are:

- ◆ No—Do not save all WebServices transaction information. (default)
- ◆ Yes—Store detailed data on all WebServices transactions, both successes and failures.

Example

```
function InitAgenda() {
    wlGlobals.SaveWSTransaction = True
}

function InitClient() {
    var BNQuoteService = new WSWebService("BNQuoteService",
    "http://www.xmethods.net/sd/2001/BNQuoteService.wsdl");
}

var retVal = BNQuoteService.getPrice("12344");
```

In this case, data on all Web Services transactions would be included in the Data Drilling report set. The Web Services transactions would be identified in the tables as `BNQuoteServices.getPrice`.

Comment

As with all `wlGlobals` configuration properties, the `SaveWSTransaction` property must be set in the `InitAgenda()` function, as is illustrated in the preceding example.

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

Browser configuration components, page 30

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

wlGlobals, page 339

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

SaveTransaction, page 227

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

script (object)

Property of Object

When working in the HTTP Protocol mode, scripts on a Web page are accessed through `script` objects that are grouped into collections of `scripts`. The `scripts` collection is a property of the following object:

document

Description

Specifies a script object in the current `document` that is interpreted by a script engine. `script` objects are grouped together within collections of `scripts`.

Syntax

The `scripts` collection includes a `length` property that reports the number of script objects within a document (read-only). To access an individual script's properties, check the `length` property of the `scripts` collection and use an index number to access the individual scripts. For example, to find out how many script objects are contained within a document, check the value of:

```
document.scripts.length
```

Access each script's properties directly using the following syntax:

```
document.scripts[index#].<scripts-property>
```

Example

```
document.scripts[1].language
```

Properties

event, page 91

id, page 143

InnerHTML, page 153

language, page 167

src, page 255

See also

Collections, page 47

document, page 79

length, page 167

search (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

link

location

Description

The search attribute string of the URL, not including the initial ? symbol (read-only string).

Example

```
"SearchFor=modems&SearchType=ExactTerm"
```

See also

link, page 169

location, page 176

Seed() (method)

Method of Object

wlRand

Description

Initialize the random number generator. Call the `Seed()` method in the `InitAgenda()` function of an agenda, using any integer as a seed.

Syntax

```
wlRand.Seed(seed)
```

Parameters

`seed`—seed integer.

Return Value

None.

Example

```
wlRand.Seed(12345)
```

See also*Num()*, page 188*Range()*, page 211*Seed()*, page 231*Select()*, page 232*wlRand*, page 355

Select

Select (object)

Property of Objects

`Select` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Denotes a list box or drop-down list. Each `Select` object represents one of the input selection fields embedded in a document. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["SELECT"]
```

Methods*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseOver()*, page 348*wlMouseUp()*, page 349*wlMultiSelect()*, page 350*wlSelect()*, page 358*wlTypeIn()*, page 366**Properties***id*, page 143*Name*, page 184*Size*, page 247**See also***Button*, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99*form*, page 102*Image*, page 148

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

Select() (method)

Method of Object

wlRand

Description

Select one element of a set at random.

Syntax

```
wlRand.Select(val1, val2, ..., valN)
```

Parameters

val1...valN—Any number of parameters itemizing the elements in a set. The parameters can be numbers, strings, or any other objects.

Return Value

The value of one of its parameters, selected at random.

Example

```
wlRand.Select(21, 57, 88, 93)
```

See also

Num(), page 188

Seed(), page 231

wlRand, page 355

Range(), page 211

Select(), page 232

selected (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

option

Description

The `selected` property has a value of `true` if this `<OPTION>` element has been selected, or `false` otherwise (read-only).

Syntax

`<NA>`

See also

location, page 176

option, page 195

selectedindex (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

element

location

Description

Indicates which of the nested `<OPTION>` elements is selected in an `element` of type `<SELECT>`. The possible values are `0`, `1`, `2`, For example, if the first `<OPTION>` element is selected, then `selectedindex = 0` (read-only). The default value is `0`.

Syntax

`<NA>`

See also

element, page 83

location, page 176

SendCounter() (function)

Description

Use this function to count the number of times an event occurs and output the value to the WebLOAD Console. Call `SendCounter()` in the main script of an Agenda.

Syntax

```
SendCounter(EventName)
```

Parameters

`EventName`—A string with the name of the event being counted.

Return Value

None

Example

```
<NA>
```

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

SendCounter(), page 234

SendMeasurement(), page 235

SendTimer(), page 236

SetTimer(), page 241

Sleep(), page 247

SynchronizationPoint(), page 281

Timing functions, page 295

SendMeasurement() (function)

Description

Use this function to assign a value to the specified statistical measurement. Call `SendMeasurement()` in the main script of an Agenda.

Syntax

```
SendMeasurement(MeasurementName, value)
```

Parameters

`MeasurementName`—A string with the name of the measurement being set.

`value`—An integer value to set.

Return Value

None

Example

```

wlHttp.Navigate("http://www.yahoo.com/")
NumberOfImagesInPage = document.images.length
SendMeasurement("NumberOfImagesInPage",NumberOfImagesInPage)

```

GUI mode

WebLOAD recommends setting measurement functions within Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a `SendMeasurement()` function to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

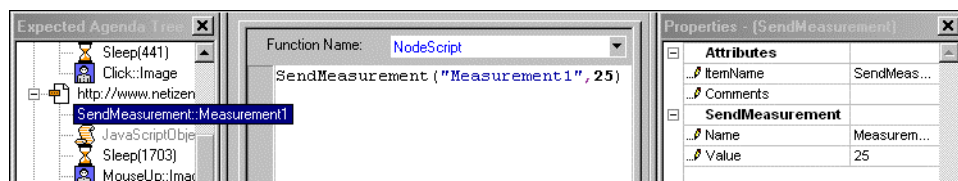


Figure 2-29: Adding a measurement node to an Agenda

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

SendCounter(), page 234

SendTimer(), page 236

Sleep(), page 247

Timing functions, page 295

SendMeasurement(), page 235

SetTimer(), page 241

SynchronizationPoint(), page 281

SendTimer() (function)

Description

Use this function to output the value of a timer to the WebLOAD Console. Call `SendTimer()` in the main script of an Agenda, immediately after any step or sequence of steps whose time you want to measure. Before the sequence of steps, you must call `SetTimer()` to zero the timer.

Syntax

```
SendTimer(TimerName)
```

Parameters

`TimerName`—A string with the name of the timer being sent to the Console.

Return Value

None

Example

```
SendTimer("Link 3 Time")
```

GUI mode

WebLOAD recommends setting timer functions within Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a `SendTimer()` function to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

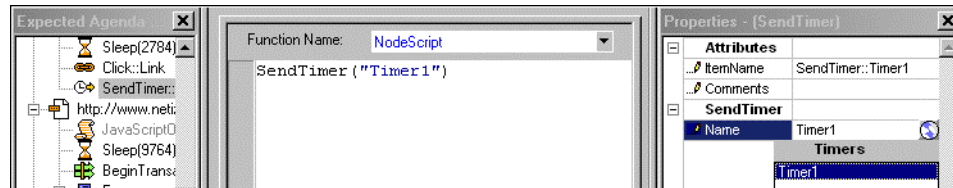


Figure 2-30: Adding a SendTimer node to an Agenda

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

SendCounter(), page 234

SendMeasurement(), page 235

SendTimer(), page 236

SetTimer(), page 241

Sleep(), page 247

SynchronizationPoint(), page 281

Timing functions, page 295

Set() (method)

Set() (addition method)

Method of Objects

wlGeneratorGlobal

wlSystemGlobal

Description

Assigns a number, Boolean, or string value to the specified shared variable. If the variable does not exist, WebLOAD will create a new variable.

Syntax

```
Set("SharedVarName", value, ScopeFlag)
```

Parameters

`SharedVarName`—The name of a shared variable to be set.

`value`—The value to be assigned to the specified variable.

`ScopeFlag`—One of two flags, `WLCurrentAgenda` or `WLAllAgendas`, signifying the scope of the shared variable.

- When used as a method of the `wlGeneratorGlobal` object:
 - ♦ The `WLCurrentAgenda` scope flag signifies variable values that you wish to share between all threads of a single Agenda, part of a single process, running on a single Load Generator.
 - ♦ The `WLAllAgendas` scope flag signifies variable values that you wish to share between all threads of one or more Agendas, common to a single spawned process, running on a single Load Generator.
- When used as a method of the `wlSystemGlobal` object:
 - ♦ The `WLCurrentAgenda` scope flag signifies variable values that you wish to share between all threads of a single Agenda, potentially shared by multiple processes, running on multiple Load Generators, system wide.
 - ♦ The `WLAllAgendas` scope flag signifies variable values that you wish to share between all threads of all Agendas, run by all processes, on all Load Generators, system-wide.

Return Value

None

Example

```
wlGeneratorGlobal.Set("MySharedCounter", 0, WLCurrentAgenda)
wlSystemGlobal.Set("MyGlobalCounter", 0, WLCurrentAgenda)
```

See also

Add(), page 21

Get(), page 110

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

User-defined global properties, page 304

wlGeneratorGlobal, page 334

wlSystemGlobal, page 363

Set() (cookie method)

Method of Object

location

wlCookie

Description

Creates a cookie.

You can set an arbitrary number of cookies in any thread. If you set more than one cookie applying to a particular domain, WebLOAD submits them all when it connects to the domain.

Syntax

```
wlCookie.Set(name, value, domain, path [, expire])
```

Parameters

name—A descriptive name identifying the type of information stored in the cookie, for example, "CUSTOMER".

value—A value for the named cookie, for example, "JOHN_SMITH".

domain—The top-level domain name to which the cookie should be submitted, for example, "www.ABCDEF.com".

path—The top-level directory path, within the specified domain, to which the cookie is submitted, for example, "/".

expire—An optional expiration timestamp of the cookie, in a format such as "Wed, 08-Apr-98 17:29:00 GMT".

Return Value

None

Comment

Set cookies within the main script of the Agenda. WebLOAD deletes all the cookies at the end of each round. If you wish to delete cookies in the middle of a round, use the `Delete()` or `ClearAll()` method.

Example

If you combine the examples used to illustrate the parameters for this method, you end up with the following:

```
wlCookie.Set("CUSTOMER", "JOHN_SMITH", "www.ABCDEF.com", "/",
            "Wed, 08-Apr-98 17:29:00 GMT")
```

Where:

- ◆ The method creates a cookie containing the data `CUSTOMER=JOHN_SMITH`. This is the data that the thread submits when it connects to a URL in the domain.
- ◆ The domain of our sample cookie is `www.ABCDEF.com/`. The thread submits the cookie when it connects to any URL in or below this domain, for example, `http://info.www.ABCDEF.com/customers/FormProcessor.exe`.

- ◆ The cookie is valid until the expiration time, which in this case is Wednesday, April 8, 1998, at 17:29 GMT.

See also

location, page 176

wlCookie, page 330

SetFailureReason() (function)

Mode

This function is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

This function enables you to specify possible reasons for a transaction failure within your transaction verification function. These reasons will also appear in the Statistics Report. The default reason for most HTTP command (Get, Post, and Head) failures is simply HTTP-Failure. Unless you specify another reason for failure, HTTP-Failure will be set automatically whenever an HTTP transaction fails on the HTTP protocol level. `SetFailureReason()` allows you to add more meaningful information to your error reports.

Syntax

```
SetFailureReason(ReasonName)
```

Parameters

`ReasonName`—A user-supplied string that identifies and categorizes the reason for this transaction instance failure.

Return Value

None.

Example

```
<NA>
```

Comment

Note that the `SetFailureReason()` function accepts a literal string as the parameter. This string identifies the cause of the failure. To get an accurate picture of different failure causes, be sure to use identification strings consistently for each failure type. For example, don't use both 'User Not Logged' *and* 'User Not LoggedIn' for the same type of failure, or your reports statistics will not be as informative. If you do not specify a specific reason for the failure, the system will register a 'General Failure', the default fail value.

See also

<i>Adding transactions</i> , page 49 in the <i>WebLOAD Programming Guide</i>	<i>BeginTransaction()</i> , page 28
<i>CreateDOM()</i> , page 59	<i>CreateTable()</i> , page 60
<i>Custom verification functions</i> , page 137 in the <i>WebLOAD Programming Guide</i>	<i>Data Drilling—WebLOAD transaction reports</i> , page 149 in the <i>WebLOAD Programming Guide</i>
<i>EndTransaction()</i> , page 85	<i>Functional Testing and Reporting</i> , page 127 in the <i>WebLOAD Programming Guide</i>
<i>ReportEvent()</i> , page 214	<i>SetFailureReason()</i> , page 239
<i>TimeoutSeverity</i> , page 293	<i>Transaction verification components</i> , page 297
<i>TransactionTime</i> , page 298	<i>Validate()</i> , page 309
<i>Verification Test Components</i> , page 315	<i>Verification Test Property List: Global and Page Level</i> , page 318
<i>VerificationFunction()</i> (user-defined), page 320	<i>wlBrowser</i> , page 325
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

SetTimer() (function)

Description

Use this function to zero a timer. Call `SetTimer()` in the main script of an Agenda, immediately before any step or sequence of steps whose time you want to measure. Be sure to zero the timer in every round of the Agenda; the timer continues running between rounds if you do not zero it.

Syntax

```
SetTimer(TimerName)
```

Parameters

`TimerName`—A string with the name of the timer being zeroed.

Return Value

None

Example

```
SetTimer("Link 3 Time")
```


GUI mode

WebLOAD recommends setting timer functions within Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a `SetTimer()` function to an Agenda Tree. The corresponding code appears in the JavaScript View pane on the right.

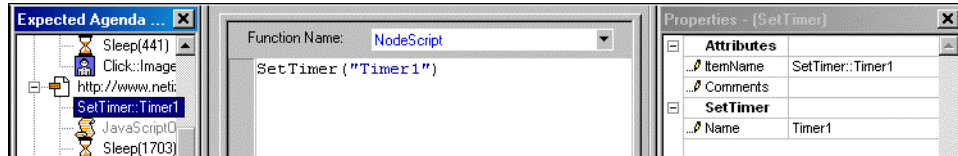


Figure 2-31: Adding a SetTimer node to an Agenda

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

SendCounter(), page 234

SendTimer(), page 236

Sleep(), page 247

Timing functions, page 295

SendMeasurement(), page 235

SetTimer(), page 241

SynchronizationPoint(), page 281

setType() (method)

Method of Object

WSComplexObject

Description

Defines the *type* of a new *WSComplexObject* object. The objects and their types correspond to the objects defined in a WSDL file.

Syntax

```
newComplexObjectName.setType("ComplexObjectName")
```

Parameters

ComplexObjectName—The object being defined, a text string.

Return Value

None.

Example

```
Worker = new WSComplexObject();
Worker.setType("Worker");
Worker.addProperty("Name", "string");
Worker.addProperty("Age", "integer");
```

Comment

New `WSComplexObject` objects must be created, and their properties defined, within the `InitClient()` function. Otherwise, a new object will be created with each iteration of the Agenda during a test session and the system will quickly run out of memory.

Web Services tools are usually based on the basic variable types, such as integer or string. Even when a complex type is used, it is usually assembled from properties that are themselves simple variables. Occasionally, an application may require complex types that are assembled from complex properties, properties that are themselves complex objects. WebLOAD supports a completely recursive system of complex object definition. The WebLOAD `WSComplexObject` object, with the `setType()`, `addProperty()`, and `setValue()` methods, systematically handles as many levels of complexity as are required to test a Web Services tool appropriately.

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

addProperty(), page 22

WSComplexObject, page 374

WSGetSimpleValue(), page 376

WSWebService(), page 379

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

setValue(), page 243

WSComplexObject(), page 375

WSWebService, page 377

setValue() (method)

Method of Object

WSComplexObject

Description

Assigns or resets a value for the properties of a `WSComplexObject` object.

Syntax

```
newComplexObject.setValue("PropertyName", "PropertyValue")
```

Parameters

`PropertyName`—Name of the property to which a value is being assigned, a text string.

PropertyValue—Value being assigned to the designated property. The value must match the type of the specified property. For example, if the property is a string, the method should be passed a text string. If the property is an integer, the method should be passed an integer.

Return Value

None.

Example

```
Worker = new WSComplexObject();
Worker.setType("Worker");
Worker.addProperty("Name", "string");
Worker.addProperty("Age", "integer");
Worker.setValue("Name", "Robert");
Worker.setValue("Age", 28);
```

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

addProperty(), page 22

WSComplexObject, page 374

WSGetSimpleValue(), page 376

WSWebService(), page 379

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

setType(), page 242

WSComplexObject(), page 375

WSWebService, page 377

SetWindow() (action)

Method of Object

wlBrowser

Description

Find the specified window and make it the active window.

Syntax

```
wlBrowser.SetWindow("Name"[, "Title"][, "Url"])
```

Parameters

Name—Name identifying the window to be set.

Title—Title of the window to be set. Optional

URL—URL of the document that is currently in the window to be set. Optional

Return Value

None

Example

```
wlBrowser.SetWindow("ShoppingBasket")
```

Comment

`SetWindow()` is added to the Agenda being recorded only if the focus is changed to a new window and at the same point an additional activity also occurs, such as taking a snapshot.

See also

Actions, page 20

Back(), page 27

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

FindObject(), page 102

Navigate(), page 186

OnClick, page 190

Refresh(), page 213

wlBrowser, page 325

wlClose(), page 329

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlSubmit(), page 362

AutoNavigate(), page 26

Dynamic Object Recognition (DOR) components, page 81

event, page 91

Forward(), page 106

ObjectProperty[], page 189

OnMouseOver, page 191

SetWindow(), page 244

wlClick(), page 326

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

wlTypeIn(), page 366

SevereErrorMessage() (function)

Description

Use this function to display a severe error message in the Log Window of the WebLOAD Console and abort the Load Generator.

Syntax

```
SevereErrorMessage (msg)
```

Parameters

`msg`—A string with a severe error message to be sent to the Console.

Return Value

None.

Example

<NA>

Comment

If you call `SevereErrorMessage()` in the main script, WebLOAD stops all activity in the Load Generator and runs the error handling functions (`OnScriptAbort()`, etc.), if they exist in the Agenda. You may also use the `wlException` object with the built-in `try()/catch()` commands to catch errors within your Agenda. For more information about error management and execution sequence options, see *Error Management*, page 104 in the *WebLOAD Programming Guide*.

GUI mode

WebLOAD recommends adding message functions to your Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a Message Node to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

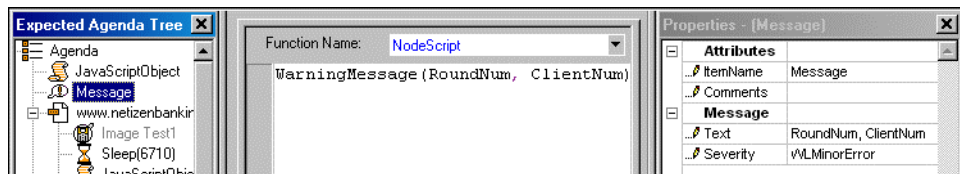


Figure 2-32: Adding a Message Node to an Agenda

See also

Error Management, page 104 in the *WebLOAD Programming Guide*

ExceptionEnabled, page 92

GetSeverity(), page 133

LiveConnect Overview, page 255 in the *WebLOAD Programming Guide*

ReportLog(), page 215

Using the IntelliSense JavaScript Editor, page 18

wlException, page 331

ErrorMessage(), page 89

GetMessage(), page 129

InfoMessage(), page 152

Message functions, page 180

SevereErrorMessage(), page 245

WarningMessage(), page 323

wlException() object constructor, page 333

Shape (property)

Property of Object

area

Description

Sets or retrieves the shape of the area object.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

area, page 24

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

Size (property)

Property of Object

element

InputFile

InputPassword

InputText

Select

Description

The size of a File, Password, Select, or Text element. When working with a Select element, determines the number of rows that will be displayed, regardless of the number of options chosen.

Syntax

<NA>

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

element, page 83

InputFile, page 159

InputPassword, page 161

InputText, page 164

Select, page 231

Sleep() (function)

Description

Pause for a specified number of milliseconds.

Syntax

```
Sleep(PauseTime)
```

Parameters

PauseTime—An integer value specifying the number of milliseconds to pause.

Return Value

None

Example

To pause for 1 second, write:

```
Sleep(1000)
```

GUI mode

WebLOAD recommends setting sleep functions within Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates a typical Sleep Node highlighted in the Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

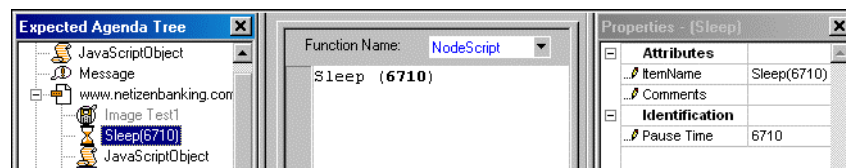


Figure 2-33: Sleep node added to an Agenda

Sleep function command lines may also be added directly to the code in a JavaScript Object within an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Users who are programming their own JavaScript Object code

within their Agenda may still take advantage of the Visual AAT GUI to simplify their programming efforts. Rather than manually type out the code for a sleep function, with the risk of making a mistake, even a trivial typo, and adding invalid code to the Agenda file, users may click on the `Sleep()` command line in the Edit | Insert | JavaScript | General menu to add a sleep command to their code. The Visual AAT automatically inserts the correct code for the sleep function into the JavaScript Object currently being edited. The user may then change the sleep time without any concerns about mistakes in the function syntax.

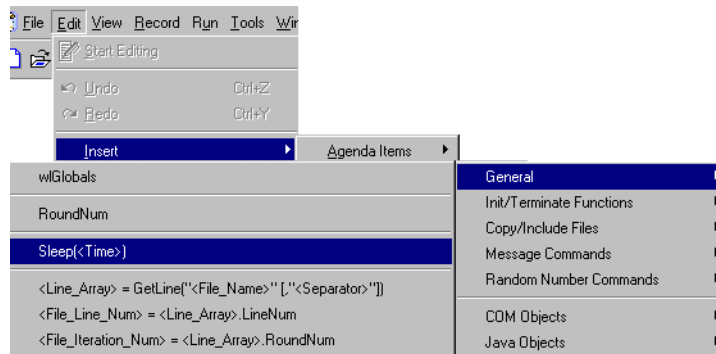


Figure 2-34: Sleep command

Comment

Specify one of three sleep options when running a test Agenda, preferably through the WebLOAD GUI using the Sleep Time Control tab from the Tools | Default options:

- ◆ Run the Agenda with the exact same sleep periods as occurred when the Agenda was originally recorded,
- ◆ Run the Agenda with no sleep periods at all,
- ◆ Run the Agenda with sleep periods of random lengths, simulating the variety of pauses that occur naturally in the real world.

WebLOAD recommends setting the sleep mode through the Visual AAT or Console GUI. Choose a setting from the Sleep Time Control tab of the Tools | Default, Current or Agenda Options dialog box, illustrated below:

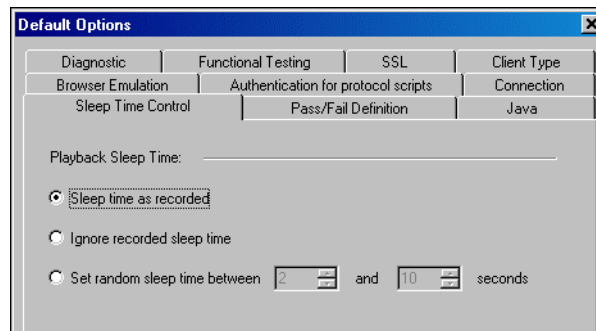


Figure 2-35: Setting the sleep mode

See also

- Adding timers*, page 26 in the *WebLOAD Programming Guide*
- DisableSleep*, page 75
- SendMeasurement()*, page 235
- SetTimer()*, page 241
- Sleeping or pausing in mid-session*, page 28 in the *WebLOAD Programming Guide*
- SleepDeviation*, page 250
- SleepRandomMin*, page 252
- Timing functions*, page 295
- wlBrowser*, page 325
- SendCounter()*, page 234
- SendTimer()*, page 236
- Sleep()*, page 247
- SleepRandomMax*, page 251
- SynchronizationPoint()*, page 281
- Using the IntelliSense JavaScript Editor*, page 18

SleepDeviation (property)

Property of Object

wlBrowser

Description

Integer that indicates the percentage by which recreated sleep periods should deviate from the original recorded time.

Example

```
wlBrowser.SleepDeviation = 10
```

Recreated sleep periods will be within a range of +/- 10% of the original recorded time.

Comment

Sleep periods during test sessions are by default kept to the length of the sleep period recorded by the user during the original recording session. If you wish to include sleep intervals but

change the time period, set `DisableSleep` to `False` and assign values to the other sleep properties as follows:

- ◆ `SleepRandomMin`—Assign random sleep interval lengths, with the minimum time period equal to this property value.
- ◆ `SleepRandomMax`—Assign random sleep interval lengths, with the maximum time period equal to this property value.
- ◆ `SleepDeviation`—Assign random sleep interval lengths, with the time period ranging between this percentage value more or less than the original recorded time period.

GUI mode

WebLOAD recommends setting the sleep mode through the Visual AAT or Console GUI. Choose a setting from the Sleep Time Control tab of the Tools | Default, Current or Agenda Options dialog box, illustrated below:

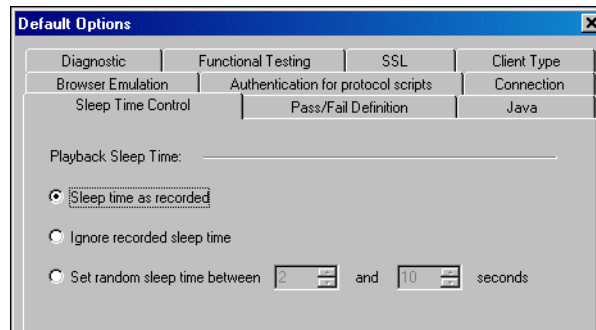


Figure 2-36: Setting the sleep mode

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

DisableSleep, page 75

Sleep(), page 247

SleepDeviation, page 250

Sleeping or pausing in mid-session, page 28 in the *WebLOAD Programming Guide*

SleepRandomMax, page 251

SleepRandomMin, page 252

wlBrowser, page 325

SleepRandomMax (property)

Property of Objects

wlBrowser

Description

Integer that indicates the maximum length of a recreated sleep period when not using the original recorded time.

Syntax

```
wlBrowser.SleepRandomMax = 5000
```

Recreated sleep periods will fall within a range whose maximum value is 5000 milliseconds.

Comment

Sleep periods during test sessions are by default kept to the length of the sleep period recorded by the user during the original recording session. If you wish to include sleep intervals but change the time period, set `DisableSleep` to `False` and assign values to the other sleep properties as follows:

- ◆ `SleepRandomMin`—Assign random sleep interval lengths, with the minimum time period equal to this property value.
- ◆ `SleepRandomMax`—Assign random sleep interval lengths, with the maximum time period equal to this property value.
- ◆ `SleepDeviation`—Assign random sleep interval lengths, with the time period ranging between this percentage value more or less than the original recorded time period.

GUI mode

WebLOAD recommends setting the sleep mode through the Visual AAT or Console GUI. Choose a setting from the Sleep Time Control tab of the Tools | Default, Current or Agenda Options dialog box, illustrated below:

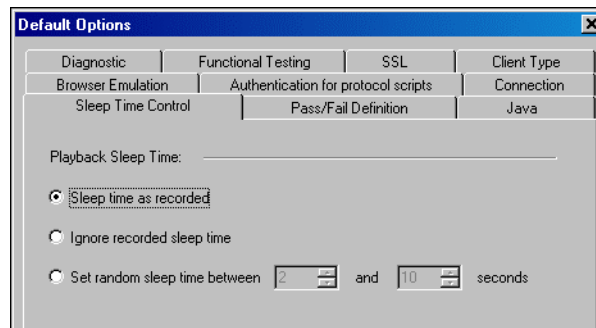


Figure 2-37: Setting the sleep mode

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

DisableSleep, page 75

Sleep(), page 247

SleepDeviation, page 250

Sleeping or pausing in mid-session, page 28 in the
WebLOAD Programming Guide

SleepRandomMax, page 251

SleepRandomMin, page 252

wlBrowser, page 325

SleepRandomMin (property)

Property of Object

wlBrowser

Description

Integer that indicates the minimum length of a recreated sleep period when not using the original recorded time.

Syntax

```
wlBrowser.SleepRandomMin = 1000
```

Recreated sleep periods will fall within a range whose minimum value is 1000 milliseconds.

Comment

Sleep periods during test sessions are by default kept to the length of the sleep period recorded by the user during the original recording session. If you wish to include sleep intervals but change the time period, set `DisableSleep` to `False` and assign values to the other sleep properties as follows:

- ◆ `SleepRandomMin`—Assign random sleep interval lengths, with the minimum time period equal to this property value.
- ◆ `SleepRandomMax`—Assign random sleep interval lengths, with the maximum time period equal to this property value.
- ◆ `SleepDeviation`—Assign random sleep interval lengths, with the time period ranging between this percentage value more or less than the original recorded time period.

GUI mode

WebLOAD recommends setting the sleep mode through the Visual AAT or Console GUI. Choose a setting from the Sleep Time Control tab of the Tools | Default, Current or Agenda Options dialog box, illustrated below:

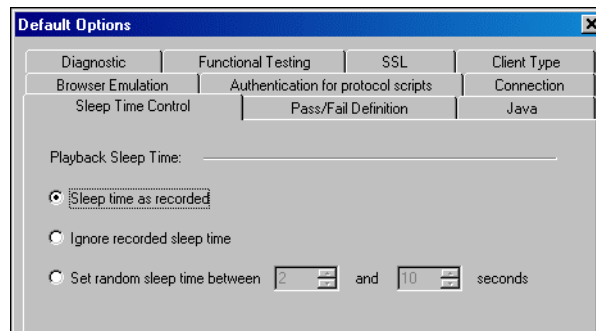


Figure 2-38: Setting the sleep mode

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

DisableSleep, page 75

Sleep(), page 247

SleepDeviation, page 250

Sleeping or pausing in mid-session, page 28 in the *WebLOAD Programming Guide*

SleepRandomMax, page 251

SleepRandomMin, page 252

wlBrowser, page 325

Span (object)

Property of Objects

Span objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Specifies an inline text container. Each Span object represents one of the `` user interface fields embedded in a document. One of the set of *UIContainer* objects. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["SPAN"]
```

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

wlMultiSelect(), page 350

wlSelect(), page 358

wlTypeIn(), page 366

Properties

ContainerTag, page 55

id, page 143

OnClick, page 190

event, page 91

InnerText, page 155

OnMouseOver, page 191

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

src (property)

Property of Object

Image

wlXmIs

script

Description

Retrieves the complete URL of the parent object, that is the URL to an external file that contains the source code or data for this image, script, or XML DOM object.

Example

```
"www.ABCDEF.com/images/logo.gif"
```

Comment

WebLOAD recommends managing images, scripts, and XML DOM objects on a Web page through the standard `document.all` collection.

See also

Collections, page 47

id, page 143

Image, page 148

InnerHTML, page 153

load(), page 170

loadXML(), page 175

load() and loadXML() method comparison, page 172

script, page 229

src, page 255

wlXmIs, page 370

XMLDocument, page 380

SSL Cipher Command Suite

Description

WebLOAD provides full SSL/TLS 1.0 protocol support through a set of a set of SSL properties for the `wlGlobals` object combined with a set of functions called the Cipher Command Suite. These SSL functions allow you to identify, enable, and disable selected SSL protocols or security levels. For a complete list of the supported SSL protocols, see Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381.

Functions

The Cipher Command Suite includes the following functions:

`SSLCipherSuiteCommand()`, page 258

`SSLDisableCipherID()`, page 263

`SSLDisableCipherName()`, page 265

`SSLEnableCipherID()`, page 266

`SSLEnableCipherName()`, page 267

`SSLGetCipherCount()`, page 268

`SSLGetCipherID()`, page 269

`SSLGetCipherInfo()`, page 270

`SSLGetCipherName()`, page 271

`SSLGetCipherStrength()`, page 272

Comment

Use the Cipher Command Suite to check or verify SSL configuration information at any point in your Agenda. However, any *changes* to an Agenda's SSL property configuration, whether through the `wlGlobals` properties or the Cipher Command Suite functions, must be made in the Agenda's *initialization functions*. Configuration changes made in the `InitAgenda()` function will affect all client threads spawned during that Agenda's test session. Configuration changes made in the `InitClient()` function will affect only individual clients. Do not make

changes to the SSL property configuration using `wlHttp` or `wlLocals` properties or in an Agenda's main body. The results will be undefined for all subsequent transactions.

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<code>SSLBitLimit</code> , page 257 (<code>wlGlobals</code> only)
<i>SSL Cipher Command Suite</i> , page 256	<code>SSLCipherSuiteCommand()</code> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<code>SSLCryptoStrength</code> , page 262 (<code>wlGlobals</code> only)
<code>SSLDisableCipherID()</code> , page 263	<code>SSLDisableCipherName()</code> , page 265
<code>SSLEnableCipherID()</code> , page 266	<code>SSLEnableCipherName()</code> , page 267
<code>SSLGetCipherCount()</code> , page 268	<code>SSLGetCipherID()</code> , page 269
<code>SSLGetCipherInfo()</code> , page 270	<code>SSLGetCipherName()</code> , page 271
<code>SSLGetCipherStrength()</code> , page 272	<code>SSLUseCache</code> , page 273
<code>SSLVersion</code> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<code>wlGlobals</code> , page 339
<code>wlHttp</code> , page 342	<code>wlLocals</code> , page 343

SSLBitLimit (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

`wlGlobals`

Description

WebLOAD provides the option of setting a limit to the maximum SSL bit length available to Virtual Clients when contacting the Server. By default, WebLOAD supports a maximum `SSLBitLimit` of 128 bits. Users may lower the `SSLBitLimit` as necessary.

You may assign an SSL bit limit value using the `wlGlobals.SSLBitLimit` property. Check the value of this property if you wish to verify the maximum cipher strength (SSL bit

limit) available for the current test session. For example, if all ciphers are enabled, then the maximum cipher strength is 128.

Note: Defining an SSL bit limit with the `SSLBitLimit` property is a low-level approach to enabling or disabling individual protocols. Even if you prefer to program property values directly rather than working through the GUI, it is usually preferable to use the `SSLCryptoStrength`, described on page 262, to define and enable cipher levels and cryptographic strengths using a higher, more categorical approach.

Syntax

```
wlGlobals.SSLBitLimit = IntegerValue
```

Example

```
wlGlobals.SSLBitLimit = 56
```

or

```
CurrentBitLimit = wlGlobals.SSLBitLimit
```

GUI mode

WebLOAD recommends setting the SSL bit limit through the Console GUI. Check SSL Bit Limit and select a value from the drop-down list on the SSL tab of the Tools | Default or Current Options dialog box, illustrated in the following figure:

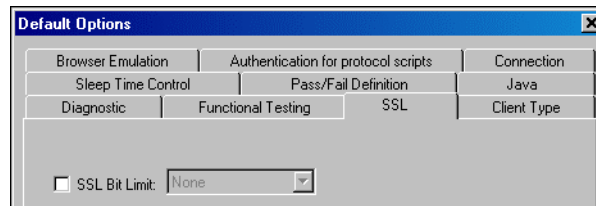


Figure 2-39: Setting SSL Bit Limit

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

Security, page 89 in the *WebLOAD Programming Guide*

`SSLBitLimit`, page 257 (wlGlobals only)

`SSLCipherCommandSuite`, page 256

`SSLCipherSuiteCommand()`, page 258

`SSLClientCertificateFile`,
`SSLClientCertificatePassword`, page 260

`SSLCryptoStrength`, page 262 (wlGlobals only)

`SSLDisableCipherID()`, page 263

`SSLDisableCipherName()`, page 265

`SSLEnableCipherID()`, page 266

`SSLEnableCipherName()`, page 267

`SSLGetCipherCount()`, page 268

`SSLGetCipherID()`, page 269

<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLCipherSuiteCommand() (function)

Description

Set the SSL configuration environment before running a test Agenda.

Syntax

```
SSLCipherSuiteCommand(SSLCipherCommand)
```

Parameters

SSLCipherCommand—One of the following commands, used to set the SSL configuration environment before running a test Agenda.

- *EnableAll*—enable all SSL protocols (default)
- *DisableAll*—disable all SSL protocols
- *ShowAll*—list all SSL protocols (provides internal information for RadView Support Diagnostics)
- *ShowEnabled*—list currently enabled SSL protocols (provides internal information for RadView Support Diagnostics)

Return Value

None.

Example

You may wish to test your application with only a single SSL protocol enabled. The easiest way to do that would be to disable all protocols, and then enable the selected protocol in the `InitAgenda()` function.

```
InitAgenda()
{
    ...
    SSLCipherSuiteCommand(DisableAll)
    SSLEnableCipherName
        ("TLS_RSA_EXPORT_WITH_RC4_40_MD5")
}
```

```
    ...
}
```

See also

- | | |
|---|---|
| <i>Browser configuration components</i> , page 30 | <i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i> |
| <i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i> | <i>SSLBitLimit</i> , page 257 (wIGlobals only) |
| <i>SSL Cipher Command Suite</i> , page 256 | <i>SSLCipherSuiteCommand()</i> , page 258 |
| <i>SSLClientCertificateFile</i> ,
<i>SSLClientCertificatePassword</i> , page 260 | <i>SSLCryptoStrength</i> , page 262 (wIGlobals only) |
| <i>SSLDisableCipherID()</i> , page 263 | <i>SSLDisableCipherName()</i> , page 265 |
| <i>SSLEnableCipherID()</i> , page 266 | <i>SSLEnableCipherName()</i> , page 267 |
| <i>SSLGetCipherCount()</i> , page 268 | <i>SSLGetCipherID()</i> , page 269 |
| <i>SSLGetCipherInfo()</i> , page 270 | <i>SSLGetCipherName()</i> , page 271 |
| <i>SSLGetCipherStrength()</i> , page 272 | <i>SSLUseCache</i> , page 273 |
| <i>SSLVersion</i> , page 275 | <i>WebLOAD-supported SSL Protocol Versions</i> , page 381 |
| <i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i> | <i>wIGlobals</i> , page 339 |
| <i>wlHttp</i> , page 342 | <i>wLocals</i> , page 343 |

SSLClientCertificateFile, SSLClientCertificatePassword (properties)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

<i>wIGlobals</i>	<i>wlHttp</i>
<i>wLocals</i>	

Description

SSL Client certificates offer a more secure method of authenticating users in an Internet commerce scenario than traditional username and password solutions. For servers that support client authentication, the server will request an identification certificate that contains information to identify the client and is signed by a recognized certificate authority. WebLOAD

supports use of SSL client certificates by supplying the certificate filename and password to the SSL server. `SSLClientCertificateFile` and `SSLClientCertificatePassword` are the filename (optionally including a directory path) and password of a certificate, which WebLOAD makes available to SSL servers. When the Agenda issues an HTTPS Get, Post, or Head command, the server can request the certificate as part of the handshake procedure. In that case, WebLOAD sends the certificate to the server, and the server can use it to authenticate the client transmission.

You may set client certificate values using the `wlGlobals` properties.

Note: You can obtain a certificate file by exporting an X.509 certificate from Microsoft Internet Explorer or Netscape Navigator. Then use the WebLOAD Certificate Wizard to convert the certificate to an ASCII (*.pem) format. For detailed instructions, see the *Recording WebLOAD Agendas* manual.

Syntax

```
wlGlobals.SSLProperty = "TextString"
```

Example

```
wlGlobals.SSLClientCertificateFile = "c:\\certs\\cert1.pem"
wlGlobals.SSLClientCertificatePassword = "topsecret"
```

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box, illustrated in *DefaultAuthentication*, page 66.

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wlGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wlGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267

<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLCryptoStrength (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

Description

Used to define the cryptographic categories to be used in the current test session. The following categories are available:

- ◆ "SSL_AllCrypto"—enable cryptography of all strengths (default).
- ◆ "SSL_StrongCryptoOnly"—enable only ciphers with strong cryptography (RSA keys greater than 512-bit and DES/EC keys greater than 40-bit)
- ◆ "SSL_ExportCryptoOnly"—enable only ciphers available for export, including only RSA keys 512-bit or weaker and DES/EC keys 40-bit or weaker.
- ◆ "SSL_ServerGatedCrypto"—verify that the communicating server is legally authorized to use strong cryptography before using stronger ciphers. Otherwise use export ciphers only.

These definitions work with your Agenda's current set of enabled ciphers. If you have enabled only certain ciphers, then setting *SSLCryptoStrength* would affect only the subset of enabled ciphers.

Example

Assume you have enabled the following four ciphers:

```
TLS_DHE_DSS_WITH_RC4_128_SHA,
SSL_RSA_WITH_DES_CBC_MD5,
```

TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA, and
 TLS_DH_anon_EXPORT_WITH_RC4_40_MD5.

If you then set SSLCryptoStrength to SSL_ExportCryptoOnly, then only the last two ciphers, TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA, and TLS_DH_anon_EXPORT_WITH_RC4_40_MD5, will be enabled.

```
InitAgenda ()
{
  ...
  SSLEnableCipherName("TLS_DHE_DSS_WITH_RC4_128_SHA")
  SSLEnableCipherName("SSL_RSA_WITH_DES_CBC_MD5")
  SSLEnableCipherName("TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA")
  SSLEnableCipherName("TLS_DH_anon_EXPORT_WITH_RC4_40_MD5")
  wlGlobals.SSLCryptoStrength="SSL_ExportCryptoOnly"
  ...
}
```

Comment

Defining a global, categorical value for SSLCryptoStrength is a high-level approach to cryptographic strength definition. This ‘smarter’ approach of selecting appropriate categories is usually preferable to the low-level approach of enabling or disabling individual protocols or defining specific SSL bit limits with the SSLBitLimit and SSLVersion properties. However, ideally SSL configuration values should be set through the WebLOAD GUI.

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wlGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wlGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLDisableCipherID() (function)

Description

Disables the specified SSL cipher for the current session.

Syntax

```
SSLDisableCipherID(CipherID)
```

Parameters

CipherID—Any of the SSL protocol ID numbers. Use the `SSLGetCipherID()` function to get the ID number associated with a specified protocol name. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

Return Value

None.

Example

Your test session may include a variety of function calls related to specific protocols. For example, you may wish to test your application with all but one SSL protocol enabled. Unfortunately, protocol names can be long and awkward. To simplify your Agenda code, you could get the ID number of a selected protocol and refer to the selected protocol by ID number for the remainder of the Agenda. The following `InitAgenda()` function fragment gets a protocol ID number and disables the selected protocol in the `InitAgenda()` function.

```
InitAgenda()
{
    ...
    MyCipherID = SSLGetCipherID
                ("TLS_RSA_EXPORT_WITH_RC4_40_MD5")
    SSLDisableCipherID(MyCipherID)
    ...
}
```

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

Security, page 89 in the *WebLOAD Programming Guide*

SSLBitLimit, page 257 (w1Globals only)

SSL Cipher Command Suite, page 256

SSLCipherSuiteCommand(), page 258

SSLClientCertificateFile,
SSLClientCertificatePassword, page 260

SSLCryptoStrength, page 262 (w1Globals only)

SSLDisableCipherID(), page 263

SSLDisableCipherName(), page 265

<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLDisableCipherName() (function)

Description

Disables the specified SSL cipher for the current session.

Syntax

```
SSLDisableCipherName (CipherName)
```

Parameters

CipherName—Any of the SSL protocol names. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

Return Value

None.

Example

You may wish to test your application with all but one SSL protocol enabled. The easiest way to do that would be to disable the selected protocol in the `InitAgenda()` function.

```
InitAgenda ()
{
    ...
    SSLDisableCipherName
        ("TLS_RSA_EXPORT_WITH_RC4_40_MD5")
    ...
}
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (<i>wlGlobals</i> only)

<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (<i>wlGlobals</i> only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLEnableCipherID() (function)

Description

Enables the specified SSL cipher for the current session.

Syntax

```
SSLEnableCipherID(CipherID)
```

Parameters

CipherID—Any of the SSL protocol ID numbers. Use the *SSLGetCipherID()* function to get the ID number associated with a specified protocol name. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

Return Value

None.

Example

Your test session may include a variety of function calls related to specific protocols. For example, you may wish to test your application with only a single SSL protocol enabled. Unfortunately, protocol names can be long and awkward. To simplify your Agenda code, you could get the ID number of a selected protocol and refer to the selected protocol by ID number for the remainder of the Agenda. The following *InitAgenda()* function fragment disables all protocols, gets a protocol ID number, and enables the selected protocol in the *InitAgenda()* function.

```
InitAgenda ()
{
```

```

...
    SSLCipherSuiteCommand(DisableAll)
    MyCipherID = SSLGetCipherID("TLS_RSA_EXPORT_WITH_RC4_40_MD5")
    SSLEnableCipherID(MyCipherID)
...
}

```

See also

- | | |
|---|---|
| <i>Browser configuration components</i> , page 30 | <i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i> |
| <i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i> | <i>SSLBitLimit</i> , page 257 (wlGlobals only) |
| <i>SSL Cipher Command Suite</i> , page 256 | <i>SSLCipherSuiteCommand()</i> , page 258 |
| <i>SSLClientCertificateFile</i> ,
<i>SSLClientCertificatePassword</i> , page 260 | <i>SSLCryptoStrength</i> , page 262 (wlGlobals only) |
| <i>SSLDisableCipherID()</i> , page 263 | <i>SSLDisableCipherName()</i> , page 265 |
| <i>SSLEnableCipherID()</i> , page 266 | <i>SSLEnableCipherName()</i> , page 267 |
| <i>SSLGetCipherCount()</i> , page 268 | <i>SSLGetCipherID()</i> , page 269 |
| <i>SSLGetCipherInfo()</i> , page 270 | <i>SSLGetCipherName()</i> , page 271 |
| <i>SSLGetCipherStrength()</i> , page 272 | <i>SSLUseCache</i> , page 273 |
| <i>SSLVersion</i> , page 275 | <i>WebLOAD-supported SSL Protocol Versions</i> , page 381 |
| <i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i> | <i>wlGlobals</i> , page 339 |
| <i>wlHttp</i> , page 342 | <i>wlLocals</i> , page 343 |

SSLEnableCipherName() (function)

Description

Enables the specified SSL cipher for the current session.

Syntax

```
SSLEnableCipherName (CipherName)
```

Parameters

CipherName—Any of the SSL protocol names. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

Return Value

None.

Example

You may wish to test your application with only a single SSL protocol enabled. The easiest way to do that would be to disable all protocols, and then enable the selected protocol in the `InitAgenda()` function:

```
InitAgenda ()
{
    ...
    SSLCipherSuiteCommand(DisableAll)
    SSLEnableCipherName("TLS_RSA_EXPORT_WITH_RC4_40_MD5")
    ...
}
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wlGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wlGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLGetCipherCount() (function)

Description

Returns an integer, the number of ciphers enabled for the current test session. While that may seem obvious if your Agenda explicitly enables two or three ciphers, it may be necessary if, for example, you have set a cipher strength limit of 40 and then wish to know how many ciphers are currently available at that limit.

Syntax

```
SSLGetCipherCount()
```

Parameters

None.

Return Value

Returns an integer representing the number of ciphers enabled for the current test session.

Example

```
CurrentCipherCount = SSLGetCipherCount()
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wGlobals</i> , page 339
<i>wHttp</i> , page 342	<i>wLocals</i> , page 343

SSLGetCipherID() (function)

Description

Returns the ID number associated with the specified cipher.

Syntax

```
SSLGetCipherID(CipherName)
```

Parameters

`CipherName`—Any of the SSL protocol names. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

Return Value

Returns the ID number associated with the specified cipher.

Example

```
MyCipherID = SSLGetCipherID("TLS_ECDH_ECDSA_WITH_RC4_128_SHA")
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wlglobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wlglobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLGetCipherInfo() (function)

Description

Prints a message on the Console with information about the specified SSL protocol.

Syntax

```
SSLGetCipherInfo(CipherName | CipherID)
```

Parameters

Accepts either one of the following parameters:

CipherName—Any of the SSL protocol names. See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a complete list of protocol names.

CipherID—Any of the SSL protocol ID numbers. Use the `SSLGetCipherID()` function to get the ID number associated with a specified protocol name.

Return Value

None.

Example

You may specify an SSL protocol using either the protocol name or the ID number. The function accepts either a string or an integer parameter, as illustrated here:

```
SSLGetCipherInfo("TLS_ECDH_ECDSA_WITH_RC4_128_SHA")
    or
SSLGetCipherInfo(2)
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wGlobals</i> , page 339
<i>wHttp</i> , page 342	<i>wLocals</i> , page 343

SSLGetCipherName() (function)

Description

Returns the name of the cipher associated with the specified ID number.

Syntax

```
SSLGetCipherName (CipherID)
```

Parameters

CipherID—Any of the SSL protocol ID numbers.

Return Value

Returns the name of the cipher associated with the specified ID number.

Example

```
MyCipherName = SSLGetCipherName (16)
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (w1Globals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (w1Globals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>w1Globals</i> , page 339
<i>w1Http</i> , page 342	<i>w1Locals</i> , page 343

SSLGetCipherStrength() (function)

Description

Returns an integer, the maximum cipher strength (SSL bit limit) available for the current test session. For example, if all ciphers are enabled, then the maximum cipher strength is 128.

Syntax

```
SSLGetCipherStrength ()
```

Parameters

None.

Return Value

Returns an integer representing the maximum available cipher strength for the current session.

Example

```
CurrentCipherStrength = SSLGetCipherStrength()
```

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wGlobals</i> , page 339
<i>wHttp</i> , page 342	<i>wLocals</i> , page 343

SSLUseCache (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wGlobals

wLocals

wHttp

Description

Enable caching of SSL decoding keys received from an SSL (HTTPS) server. The value of `SSLUseCache` may be:

- ◆ No—Disable caching.
- ◆ Yes—Enable caching. (default)

A "Yes" value means that WebLOAD receives the key only on the first SSL connection in each round. In subsequent connections, WebLOAD retrieves the key from the cache.

Assign a "Yes" value to reduce transmission time during SSL communication. Assign a "No" value if you want to measure the transmission time of the decoding key in the WebLOAD performance statistics for each SSL connection.

If you enable caching, you can clear the cache at any time by calling the `wlHttp.ClearSSLCache()` method. The cache is automatically cleared at the end of each round.

Example

<NA>

GUI mode

WebLOAD recommends enabling or disabling the SSL cache through the Console GUI. Enable caching for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

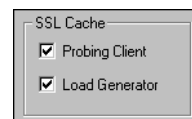


Figure 2-40: Enabling SSL Cache for Load Generator

Comment

To clear the SSL cache when working in HTTP Protocol mode, set the `ClearSSLCache()` property, described on page 42.

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

Security, page 89 in the *WebLOAD Programming Guide*

`SSLBitLimit`, page 257 (wlGlobals only)

SSL Cipher Command Suite, page 256

`SSLCipherSuiteCommand()`, page 258

`SSLClientCertificateFile`,

`SSLCryptoStrength`, page 262 (wlGlobals)

<i>SSLClientCertificatePassword</i> , page 260	only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339
<i>wlHttp</i> , page 342	<i>wlLocals</i> , page 343

SSLVersion (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlHttp

wlLocals

Description

The SSL version that WebLOAD should use for the current test session. The possible values of `wlGlobals.SSLVersion` are:

- ◆ `SSL_Version_Undetermined`—(Default) WebLOAD can use any SSL protocol version, allowing the broadest interoperability with other SSL installations. WebLOAD sends initial messages using SSL 2.0, then attempts to negotiate up to SSL 3.0. If the peer requests SSL 2.0 communications, SSL 2.0 is used for further communication.
Note: *WebLOAD does not recommend changing the default value.*
- ◆ `SSL_Version_3_0_With_2_0_Hello`—WebLOAD sends initial messages using SSL 2.0, but all subsequent communication must be through SSL 3.0 only. Otherwise the connection will fail with a meaningful error message.
- ◆ `TLS_Version_1.0_With_2_0_Hello`—WebLOAD sends initial messages using SSL 2.0, but all subsequent communication must be through TLS 1.0 only. Otherwise the connection will fail with a meaningful error message.

- ◆ `SSL_Version_3_0_Only`—All communication is by SSL 3.0 only. If the peer does not support SSL 3.0, the handshake fails without a meaningful indication of why it failed. Use this option for highest security when working with peers that definitely support SSL 3.0.
- ◆ `TLS_Version_1_0_Only`—All communication is by TLS 1.0 only. If the peer does not support TLS 1.0, the handshake fails without a meaningful indication of why it failed. Use this option for highest security when working with peers that definitely support TLS 1.0.
- ◆ `SSL_Version_3_0`—WebLOAD sends initial messages using SSL 3.0. If the peer requests SSL 2.0 communications, SSL 2.0 is used for further communication.
- ◆ `SSL_Version_2_0`—WebLOAD sends initial messages and all further communication using SSL 2.0. This option is not recommended other than for testing, because SSL 3.0 is more functional and secure than SSL 2.0.
- ◆ `TLS_Version_1_0`—WebLOAD sends initial messages using TLS 1.0. If the peer requests SSL 3.0 communications, SSL 3.0 is used for further communication.

To connect to a server using any of the SSL options, include `https://` in the URL.

Example

```
wlGlobals.SSLVersion = "SSL_Version_3_0_Only"
wlGlobals.Url = https://www.ABCDEF.com
```

See Appendix A, *WebLOAD-supported SSL Protocol Versions*, page 381, for a table illustrating all the Client/Server SSL version handshake combination possibilities and a complete list of SSL/TLS protocol names.

See also

<i>Browser configuration components</i> , page 30	<i>Rules of scope for local and global variables</i> , page 111 in the <i>WebLOAD Programming Guide</i>
<i>Security</i> , page 89 in the <i>WebLOAD Programming Guide</i>	<i>SSLBitLimit</i> , page 257 (wlGlobals only)
<i>SSL Cipher Command Suite</i> , page 256	<i>SSLCipherSuiteCommand()</i> , page 258
<i>SSLClientCertificateFile</i> , <i>SSLClientCertificatePassword</i> , page 260	<i>SSLCryptoStrength</i> , page 262 (wlGlobals only)
<i>SSLDisableCipherID()</i> , page 263	<i>SSLDisableCipherName()</i> , page 265
<i>SSLEnableCipherID()</i> , page 266	<i>SSLEnableCipherName()</i> , page 267
<i>SSLGetCipherCount()</i> , page 268	<i>SSLGetCipherID()</i> , page 269
<i>SSLGetCipherInfo()</i> , page 270	<i>SSLGetCipherName()</i> , page 271
<i>SSLGetCipherStrength()</i> , page 272	<i>SSLUseCache</i> , page 273
<i>SSLVersion</i> , page 275	<i>WebLOAD-supported SSL Protocol Versions</i> , page 381
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	<i>wlGlobals</i> , page 339

wlHttp, page 342

wlLocals, page 343

state (property)

Property of Object

wlMediaPlayer

Description

Retrieves a value indicating the state of the Media Player operation, such as connecting, buffering, playing, etc. (read-only long number)

Syntax

```
MyMediaPlayerObject.state
```

Example

<NA>

See also

bitrate, page 30

connectionBandwidth, page 53

currentPosition, page 62

currentStreamName, page 62

duration, page 80

fileName, page 100

OpenStream(), page 194

Pause(), page 200

Play(), page 200

Resume(), page 220

state, page 277

Stop(), page 278

type, page 300

wlMediaPlayer, page 344

Stop() (method)

Method of Object

wlMediaPlayer

Description

This method stops playing without changing the logical stream position.

Syntax

```
MyMediaPlayerObject.Stop()
```

Parameters

None

Return Value

None.

Example

<NA>

See also*bitrate*, page 30*currentPosition*, page 62*duration*, page 80*OpenStream()*, page 194*Play()*, page 200*state*, page 277*type*, page 300*connectionBandwidth*, page 53*currentStreamName*, page 62*fileName*, page 100*Pause()*, page 200*Resume()*, page 220*Stop()*, page 278*wlMediaPlayer*, page 344

string (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*title***Description**

Stores the document title in a text string.

Syntax

<NA>

See also*title*, page 296

Submit (object)

Property of Objects

Submit objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

One of the form input controls, where `input type = submit`. Each Submit object represents a button that, when clicked, submits the form. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["SUBMIT"]
```

Methods

wlClick(), page 326

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

Properties

event, page 91

InnerText, page 155

OnClick, page 190

value, page 310

id, page 143

Name, page 184

title, page 296

See also

Button, page 34

Collections, page 47

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

Select, page 231

TextArea, page 292

Checkbox, page 38

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

Radiobutton, page 210

text, page 291

UIContainer, page 301

SyncDOM() (method)

Method of Object

wlBrowser

Description

Saves a complete snapshot of the current state of the DOM into the actual database after any navigation activity or changes in window content or focus. Session snapshots are created automatically during recording and testing sessions following every navigation action or change of focus. The method description and sample code fragment included here are intended only to clarify the code that you may see in the JavaScript Agendas created for you automatically by the Visual AAT.

Syntax

```
wlBrowser.SyncDOM(IndexNumber)
```

Parameters

IndexNumber—Index number of the snapshot in the *actual* database.

Return Value

None.

Example

```
wlBrowser.Navigate("http://www.abc.com")
wlBrowser.SyncDOM(1)

// this action completes a navigation action and a snapshot
wlBrowser.ObjectProperty["Image_Url"]="http://www.abc.com/images/prod.gif"
wlBrowser.ObjectProperty["Location"] = "bottom#1.left#0"
wlBrowser.ObjectProperty["Url"] = "http://www.abc.com/products.htm"
link1 = wlBrowser.FindObject(FT_LINK, 1)
wlBrowser.ExpectNavigation("http://www.abc.com/images/prod.gif")
link1.wlClick()
wlBrowser.SyncDOM(2)
...

// this action does not complete a navigation event,
// so does not automatically create a snapshot
wlBrowser.ObjectProperty["Image_Url"]="http://www.abc.com/images/prod.gif"
wlBrowser.ObjectProperty["Location"] = "bottom#1.left#0"
wlBrowser.ObjectProperty["Url"] = "http://www.abc.com/products.htm"
Image1 = wlBrowser.FindObject(FT_IMAGE, 2)
Image1.wlMouseOver()
...

// this action does not normally trigger an automatic snapshot,
// but the user inserted one through the WebLOAD GUI
```

```

wlBrowser.ObjectProperty["Location"] = "bottom#1.left#0"
wlBrowser.ObjectProperty["Url"] = "http://www.abc.com/products.htm"
Image2 = wlBrowser.FindObject(FT_IMAGE, 3)
Image2.wlMouseOver()
wlBrowser.SyncDOM(3)
...

// here the user selected a different window and changed focus
// to that window, again triggering an automatic snapshot
wlBrowser.SetWindow("ProductList")
wlBrowser.SyncDOM(4)

```

See also

wlBrowser, page 325

SynchronizationPoint() (function)

Description

WebLOAD provides Synchronization Points to coordinate the actions of multiple Virtual Clients. A Synchronization Point is a meeting place where Virtual Clients wait before continuing with an Agenda. When one Virtual Client arrives at a Synchronization Point, WebLOAD holds the Client at the point until all the other Virtual Clients arrive. When all the Virtual Clients have arrived, they are all released at once to perform the next action in the Agenda simultaneously. For more information on Synchronization Points, see *SynchronizationPoint()*, page 281 in the *WebLOAD Programming Guide*.

Syntax

```
SynchronizationPoint([timeout])
```

Parameters

timeout—An optional integer value that sets the number of milliseconds that WebLOAD will wait for all of the Virtual Clients to arrive at the Synchronization Point. The timeout parameter is a safety mechanism that prevents an infinite wait if any of the Virtual Clients does not arrive at the Synchronization Point for any reason. Once the timeout period expires, WebLOAD releases the rest of the Virtual Clients. By default, there is no timeout value. WebLOAD will wait an infinite amount of time for all Virtual Clients to arrive. Setting a timeout value is important to ensure that the test session will not ‘hang’ indefinitely in case of error.

Return Value

SynchronizationPoint() functions return one of the following values. These values may be checked during the Agenda runtime.

- ◆ **WLSuccess**—synchronization succeeded. All Virtual Clients arrived at the Synchronization Point and were released together.

- ◆ `WLLoadChanged`—synchronization failed. A change in the Load Size was detected while Virtual Clients were being held at the Synchronization Point. All Virtual clients were released.
- ◆ `WLTimeout`—synchronization failed. The timeout expired before all Virtual Clients arrived at the Synchronization Point. All Virtual Clients were released.
- ◆ `WLError`—synchronization failed. Invalid timeout value. All Virtual Clients were released.

Example

To test a Web application with all the Virtual Clients performing a particular Post operation simultaneously, add a Synchronization Point as follows. The various return values are highlighted:

The following Agenda fragment illustrates a typical use of synchronization points. To test a Web application with all the Virtual Clients performing a particular Post operation simultaneously, add a Synchronization Point as follows. The various return values are highlighted:

```
wlHttp.Get ("url")
...
SP = SynchronizationPoint(10000)
if (SP == WLLoadChanged)
{
  InfoMessage("Synchronization failed, Load Size changed")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
if (SP == WLTimeout)
{
  InfoMessage("Synchronization failed, Timeout expired")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
if (SP == WLError)
{
  InfoMessage("Synchronization failed")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
if (SP == WLSuccess)
{
  InfoMessage("Synchronization succeeded")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
wlHttp.Post (url)
```

GUI mode

WebLOAD recommends setting synchronization functions within Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates a typical Synchronization Point highlighted in the Agenda Tree. The JavaScript code line that corresponds to the Synchronization Point in the Agenda Tree appears in the JavaScript View pane in the center, and the property list for that Synchronization Point appears in the Properties pane to the right.

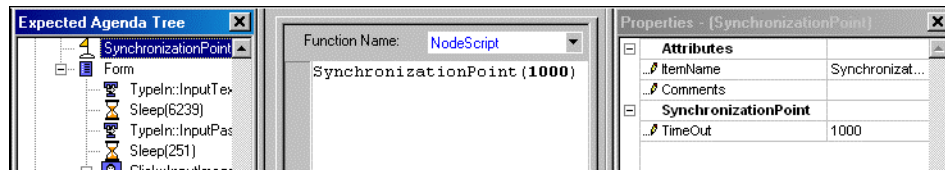


Figure 2-41: Adding a Synchronization Point to an Agenda

Comment

If there is a change in the Load Size (scheduled or unscheduled) or if any WebLOAD component is paused or stopped during the test session, all Synchronization Points are disabled.

Only client threads running within a single spawned process, on the same Load Generator, are able to share user-defined global variables and synchronization points. So if, for example, you have spawning set to 100 and you are running a total of 300 threads, realize that you are actually running three spawned processes on three separate Load Generators. You will therefore only be able to synchronize 100 client threads at a time, and not all 300.

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

SendCounter(), page 234

SendMeasurement(), page 235

SendTimer(), page 236

SetTimer(), page 241

Sleep(), page 247

SynchronizationPoint(), page 281

Timing functions, page 295

table (object)

Description

Each `table` object stores Web page material that has been organized into rows and columns. The `table` objects are accessed through collections of `document.all` `table` objects. (Individual items within a table are accessed through the `TableCell`.)

Syntax

```
document.all.tags("TABLE")[index#].tableProperty
```

Example

```
myFirstTableObject = document.all.tags("TABLE")[0]
Access specific rows or columns or table cells the same way. For example:
document.all.tags("TABLE")[0].rows[1].rowProperty
OR (in a shorter format)
myFirstTableObject.rows[1].rowProperty
```

Properties

id, page 143

Comment

WebLOAD recommends working with tables on a Web page through the `table` object rather than the `wlTables` collection.

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Collections, page 47

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

FindObject(), page 102

length, page 167

table, page 284

TableCell, page 285

wlTable, page 364

TableCell (object)

Description

Each `TableCell` object stores one component of the data found on a table. `TableCells` are used to represent both a table's header (TH) and data (TD) cells. (The complete collection of table cells is represented by the `table` object.)

Syntax

<NA>

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

Properties*id*, page 143*event*, page 91*OnMouseOver*, page 191*InnerText*, page 155*OnClick*, page 190**Comment**

WebLOAD recommends working with tables on a Web page through the `table` object rather than the `wlTables` collection.

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also*Collections*, page 47*FindObject()*, page 102*table*, page 284*wlTable*, page 364*Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide**length*, page 167*TableCell*, page 285

TableCompare (object)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

If you choose to work with the manual-mode AAT rather than the Visual AAT, WebLOAD provides a Table Compare Definition Wizard to simplify the sometimes-complex task of comparing Web site table entries. To maintain a long-term record of the Table Compare results, WebLOAD also provides a `TableCompare` object that may be added manually to your Agenda. Use the `TableCompare` object to record any failures or discrepancies as `TableCompareEvent` entries in the Data Drilling report, described in *Data Drilling—WebLOAD transaction reports*, page 149 in the *WebLOAD Programming Guide*.

Syntax

<NA>

Comment

`TableCompare` objects must not be global. Each new object must be declared locally within each thread. Generally, you should create new `TableCompare` objects only in the `InitClient()` function of an Agenda, not in the main script. If a statement in the main script creates an object, a new object is created each time the statement is executed. If WebLOAD repeats the main script many times, a large number of objects may be created and the system may run out of memory.

A complete implementation of the `TableCompare` object, with all its properties and methods, is provided in the WebLOAD include directory in the `tablecompare.js` file. For a detailed discussion of the various issues and factors that should be considered and decisions that must be made when defining and working with a `TableCompare` object, see *Table Compare Definition Wizard*, page 142 in the *WebLOAD Programming Guide*.

Methods

`Compare()`, page 49

`Prepare()`, page 205

`TableCompare()`, page 287

Properties

`CompareColumns`, page 51

`CompareRows`, page 52

`Details`, page 72

`MatchBy`, page 178

`ReportUnexpectedRows`, page 216

Comment

The `TableCompare` object is used with the `TableCompare Wizard`, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

`cell`, page 35 (`wlTable` and `row` property)

`cellIndex`, page 37 (`cell` property)

`Collections`, page 47

`cols`, page 48 (`wlTable` property)

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

`FindObject()`, page 102

`id`, page 143 (`wlTable` property)

`InnerHTML`, page 153 (`cell` property)

`InnerText`, page 155 (`cell` property)

`length`, page 167

`row`, page 223 (`wlTable` property)

`rowIndex`, page 225 (`row` property)

`table`, page 284

`TableCell`, page 285

`TableCompare`, page 286

`tagName`, page 289 (`cell` property)

`wlTable`, page 364

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

TableCompare() (constructor)

Method of Object

TableCompare

Description

Creates a new `TableCompare` object.

Syntax

Recommended:

```
TableCompare(ExpectedTable)
```

Alternate:

```
TableCompare(ExpectedTable, ActualTable)
```

Parameters

`ExpectedTable`—Pointer to an HTML table, the table being used as a ‘standard’, against which another table will be compared.

`ActualTable`—Pointer to an HTML table, the table you wish to compare to the ‘standard’ table.

Return Value

Returns a new `TableCompare` object.

Example

Recommended:

```
a=new TableCompare(ExpectedTable)
```

Alternate:

```
b=new TableCompare(ExpectedTable, ActualTable)
```

Comment

The `TableCompare` object is used with the `TableCompare Wizard`, which is only available to users of the manual AAT Agenda recording program. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection.

See also

cell, page 35 (`wlTable` and `row` property)

cellIndex, page 37 (`cell` property)

Collections, page 47

cols, page 48 (`wlTable` property)

Compare(), page 49

CompareColumns, page 51

<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<i>wlTable</i> property)	<i>InnerHTML</i> , page 153 (<i>cell</i> property)
<i>InnerText</i> , page 155 (<i>cell</i> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<i>wlTable</i> property)	<i>rowIndex</i> , page 225 (<i>row</i> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<i>cell</i> property)
<i>wlTable</i> , page 364	<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>

tagName (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

cell

Description

A string containing the cell type, either <TD> or <TH>.

Syntax

Use the following syntax to check a particular table cell type:

```
document.wlTables.myTable.cells[index#].tagName
```

Example

```
<NA>
```

Comment

The *tagName* property is a member of the *wlTables* family of table, row, and cell objects. WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

<i>cell</i> , page 35 (<code>wlTable</code> and <code>row</code> property)	<i>cellIndex</i> , page 37 (<code>cell</code> property)
<i>Collections</i> , page 47	<i>cols</i> , page 48 (<code>wlTable</code> property)
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>id</i> , page 143 (<code>wlTable</code> property)	<i>InnerHTML</i> , page 153 (<code>cell</code> property)
<i>InnerText</i> , page 155 (<code>cell</code> property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>row</i> , page 223 (<code>wlTable</code> property)	<i>rowIndex</i> , page 225 (<code>row</code> property)
<i>TableCompare</i> , page 286	<i>tagName</i> , page 289 (<code>cell</code> property)
<i>wlTable</i> , page 364	<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>

target (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object*form**link***Description**

The name of the window or frame into which the form or link should be downloaded (read-only string).

Syntax

<NA>

Example

In the following code fragment:

```
<A HREF="newpage.htm" TARGET="_top">
Go to New Page.
</A>
```

The `target` property would equal “_top” and the link will load the page into the top frame of the current frameset.

Comment

Note that while `link` and `location` objects share most of their properties, the `target` property is used by the `link` object only and is not accessed by the `location` object.

The `form.target` and `link.target` properties identify the most recent, immediate location of the target frame using the name string or keyword that was assigned to that frame. Compare this to the `wlHttp.wlTarget` property of a transaction, which uses the WebLOAD shorthand notation, described in *Identifying frame locations*, page 224 in the *WebLOAD Programming Guide*, to store the complete path of the frame, from the root window of the Web page. The last field of the `wlHttp.wlTarget` string is the target name stored in the `form.target` and `link.target` properties.

See also

form, page 102

link, page 169

wlTarget, page 366

text (property)

Property of Objects

Text objects on a Web page are accessed through the `document.all` collection of the standard DOM structure. Text objects are also properties of the following object:

option

Description

Each `Text` object represents one of the text fields embedded in a document. When a property of the `option` object, holds the text contained within the `<OPTION>` element. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["TEXT"]
```

Methods

wlClick(), page 326

wlMouseDown(), page 347

wlMouseOver(), page 348

wlMouseUp(), page 349

wlMultiSelect(), page 350

wlSelect(), page 358

wlTypeIn(), page 366

Properties*id*, page 143*Name*, page 184*Size*, page 247**Comment**

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also*Button*, page 34*Checkbox*, page 38*Collections*, page 47*File*, page 99*form*, page 102*Image*, page 148*InputButton*, page 157*InputCheckbox*, page 158*InputFile*, page 159*InputImage*, page 160*InputRadio*, page 162*InputText*, page 164*length*, page 167*Radiobutton*, page 210*Select*, page 231*text*, page 291*TextArea*, page 292*UIContainer*, page 301

TextArea (object)

Property of Objects

`TextArea` objects on a Web page are accessed through the `document.all` collection of the standard DOM structure.

Description

Each `TextArea` object specifies a multiline text input control, representing one of the text fields embedded in a document. (Compare to the *element* object, which stores the parsed data for a single HTML form element, where the element may be any one of a variety of types, and the *form* object, which stores the parsed data for an entire HTML form.)

Syntax

```
document.all.tags["TEXTAREA"]
```

Methods*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseOver()*, page 348*wlMouseUp()*, page 349

wlMultiSelect(), page 350

wlSelect(), page 358

wlTypeIn(), page 366

Properties

id, page 143

MaxLength, page 180

Name, page 184

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Checkbox, page 38

Collections, page 47

File, page 99

form, page 102

Image, page 148

InputButton, page 157

InputCheckbox, page 158

InputFile, page 159

InputImage, page 160

InputRadio, page 162

InputText, page 164

length, page 167

Radiobutton, page 210

Select, page 231

text, page 291

TextArea, page 292

UIContainer, page 301

TimeoutSeverity (property)

Property of Objects

wlBrowser

Description

When conducting Page Verification tests, `TimeoutSeverity` stores the error level to be triggered if the full set of verification tests requested for the current page are not completed within the specified time limit.

GUI mode

WebLOAD recommends setting page verification severity levels through the Visual AAT or Console GUI. Check Page Verification and select an error severity level from the drop-down box in the Functional Testing tab of the Tools | Default or Current Project Options dialog box, illustrated in the following figure:

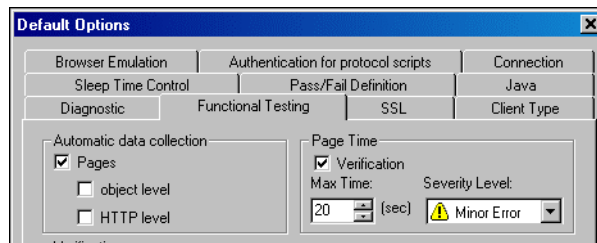


Figure 2-42: Setting Page Verification error thresholds

Syntax

You may also assign a severity level using the `TimeoutSeverity` property.

```
wlBrowser.TimeoutSeverity = ErrorFlag
```

The following error codes are available:

- ◆ `WLSuccess`—the transaction terminated successfully.
- ◆ `WLMinorError`—this specific transaction failed, but the test session may continue as usual. The Agenda displays a warning message in the Log window and continues execution from the next statement.
- ◆ `WLError`—this specific transaction failed and the current test round was aborted. The Agenda displays an error message in the Log window and begins a new round.
- ◆ `WLSevereError`—this specific transaction failed and the test session must be stopped completely. The Agenda displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

```
wlBrowser.TimeoutSeverity = WLError.
```

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

VerificationFunction() (user-defined), page 320 *wlBrowser*, page 325

Working in HTTP Protocol Mode, page 213 in the
WebLOAD Programming Guide

Timing functions

Description

The timer functions let you time or synchronize any operation or group of user activities in an Agenda, such as a navigation or mouse click, and send the time statistics to the WebLOAD Console.

Example

The following Agenda connects to the home Web page of company. On every fifth round, the Agenda also connects to a second Web page. The Agenda uses different timers to measure the time for each connection. Note that this Agenda fragment contains a main script only.

WebLOAD reports three time statistics:

- ◆ The round time, which includes both connections.
- ◆ Page 1 Time, reported in every round for the first connection only.
- ◆ Page 2 Time, reported in every fifth round for the second connection only.

```
SetTimer("Page 1 Time")
wlHttp.Get("http://www.ABCDEF.com")
SendTimer("Page 1 Time")
if (RoundNum%5 == 0) {
    SetTimer("Page 2 Time")
    wlHttp.Get("http://www.ABCDEF.com/product_info.html")
    SendTimer("Page 2 Time")
}
```

Functions

The set of timer functions includes the following:

SendCounter(), page 234

SendMeasurement(), page 235

SendTimer(), page 236

SetTimer(), page 241

Sleep(), page 247

SynchronizationPoint(), page 281

See also

Adding timers, page 26 in the *WebLOAD Programming Guide*

Timing functions, page 295

title (property)

Property of Objects

<i>Button</i>	<i>Div</i>
<i>document</i>	<i>element</i>
<i>frames</i>	<i>Image</i>
<i>InputButton</i>	<i>InputCheckbox</i>
<i>link</i>	<i>location</i>
<i>script</i>	<i>Span</i>
<i>TableCell</i>	<i>UIContainer</i>
<i>window</i>	

Description

Stores the title value associated with the parent object.

When working with `document` objects, a `title` property is an object that contains the document title, stored as a text string. When working with `window` objects, the title is extracted from the document inside the window. `title` objects are local to a single thread. You cannot create new `title` objects using the JavaScript `new` operator, but you can access a document title through the properties and methods of the standard DOM objects. The properties of `title` are read-only.

When working with `element`, `link`, or `location` objects, a `title` property contains the title of the parent `Button`, `Checkbox`, `Reset`, or `Submit` element or `link` object. May be used as a tooltip string. When working with `document` objects, a `title` property is an object that contains the document title, stored as a text string. When working with `window` objects, the title is extracted from the document inside the window.

Syntax

Document object:

Access the title's properties directly using the following syntax:

```
document.title.<titleproperty>
```

Example

Document object:

```
CurrentDocumentTitle = document.title.string
```

Properties

Document object:

string, page 278

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Button, page 34

Collections, page 47

element, page 83

form, page 102

InputButton, page 157

InputFile, page 159

InputRadio, page 162

length, page 167

location, page 176

Select, page 231

TextArea, page 292

window, page 324

Checkbox, page 38

document, page 79

File, page 99

Image, page 148

InputCheckbox, page 158

InputImage, page 160

InputText, page 164

link, page 169

Radiobutton, page 210

text, page 291

UIContainer, page 301

Transaction verification components

Description

Customized transaction verification functions are created out of the following components:

BeginTransaction(), page 28

CreateTable(), page 60

ReportEvent(), page 214

VerificationFunction() (*user-defined*), page 320

CreateDOM(), page 59

EndTransaction(), page 85

SetFailureReason(), page 239

For a more complete explanation and examples of functional testing and transaction verification, see *Functional Testing and Reporting*, beginning on page 127 in the *WebLOAD Programming Guide*.

See also

Adding transactions, page 49 in the *WebLOAD*

Custom verification functions, page 137 in the

Programming Guide

Data Drilling—WebLOAD transaction reports,
page 149 in the *WebLOAD Programming Guide*

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

wlBrowser, page 325

WebLOAD Programming Guide

Functional Testing and Reporting, page 127 in
the *WebLOAD Programming Guide*

Transaction verification components, page 297

Validate(), page 309

*Verification Test Property List: Global and Page
Level,* page 318

Working in HTTP Protocol Mode, page 213 in
the *WebLOAD Programming Guide*

TransactionTime (property)

Property of Objects

wlBrowser

Description

Assign a timeout value using the `wlBrowser.TransactionTime` property. When conducting Page Verification tests, `TransactionTime` stores the maximum amount of time that should be needed to complete the full set of verification tests requested for the current page.

GUI mode

WebLOAD recommends setting page verification timeout values through the Visual AAT or Console GUI. Check Page Verification and enter a maximum number of seconds in the Functional Testing tab of the Tools | Default or Current Project Options dialog box, illustrated in the following figure:

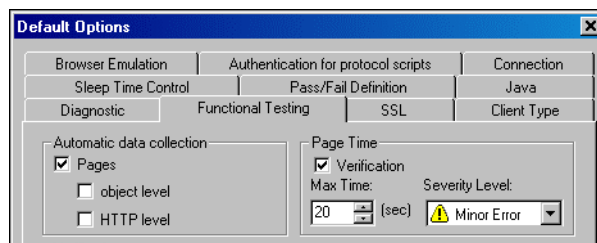


Figure 2-43: Setting Page Verification error thresholds

Syntax

You may also assign a timeout value using the `TransactionTime` property.

```
wlBrowser.TransactionTime = TimeValue
```


Example

The value assigned to `TransactionTime` may be written in either string or integer format, where the integer represents the number of milliseconds to wait and the string represents the decimal fraction of a whole second. Therefore, the following two lines are equivalent, both setting `TransactionTime` to one millisecond:

```
wlBrowser.TransactionTime = 1
```

or

```
wlBrowser.TransactionTime = "0.001"
```

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

type (property)

Property of Objects

element

wlBrowser

wlHttp.Data

wlMediaPlayer

form

wlHttp

wlHttp.DataFile

Description

This property is a string that holds the ‘type’ of the parent object.

If the parent is a form `element` object, then `type` holds the HTML type attribute of the form element. For example, an `<INPUT>` element can have a type of "TEXT", "CHECKBOX", or "RADIO". Certain HTML form elements, such as `<SELECT>` do not have a type attribute. In that case, `element.type` is the element tag itself, for example "SELECT".

If the parent is a `wlBrowser` object, then `Type` holds the HTML type attribute value that the WebLOAD DOM has stored for the current object, such as "A" for links or anchors, "AREA" for client-side image maps, "Map" for server-side image maps, "FORM" for forms, "META" for meta objects, and "NO_OBJ" when there is no value set. Note that the `Type` value *does not* change. Even when working with dynamic HTML, the type of a specific object remains the same through all subsequent transactions with that object.

If the parent is a `wlHttp.Data` or `wlHttp.DataFile` object, then `Type` holds the MIME type of the string or form data being submitted through an HTTP Post command.

If the parent is a `wlMediaPlayer` object, then `type` retrieves the major type of the stream. (audio/video/script) (read-only string)

Syntax

element:

When working with `element` objects, use the lowercase form:

```
<NA>
```

wlBrowser:

When working with `wlBrowser` objects, use the uppercase form:

```
wlHttp.Type = "A"
```

wlHttp.Data:

When working with `wlHttp.Data` objects, use the uppercase form:

```
wlHttp.Data.Type = "application/x-www-form-urlencoded"
```

wlMediaPlayer

When working with `wlMediaPlayer` objects, use the lowercase form:

```
MyMediaPlayerObject.type
```

Comment

Notice that the `Type` property for `wlHttp.Data` and `wlHttp.DataFile` objects is written in uppercase.

See also

bitrate, page 30

connectionBandwidth, page 53

currentPosition, page 62

currentStreamName, page 62

<i>Data</i> , page 63	<i>DataFile</i> , page 65
<i>duration</i> , page 80	<i>element</i> , page 83
<i>Erase</i> , page 87	<i>fileName</i> , page 100
<i>form</i> , page 102	<i>FormData</i> , page 104
<i>Get()</i> , page 110	<i>Header</i> , page 138
<i>OpenStream()</i> , page 194	<i>Pause()</i> , page 200
<i>Play()</i> , page 200	<i>Post()</i> , page 202
<i>Resume()</i> , page 220	<i>state</i> , page 277
<i>Stop()</i> , page 278	<i>type</i> , page 300
<i>value</i> , page 310	<i>wlBrowser</i> , page 325
<i>wlClear()</i> , page 328	<i>wlHttp</i> , page 342
<i>wlMediaPlayer</i> , page 344	<i>wlMediaPlayer()</i> , page 345

UIContainer (object)

Description

The `UIContainer` object is a generic term used by WebLOAD to represent any of the set of objects that are user interface containers, such as `<DIV>` or ``. `UIContainer` objects are accessed through collections of `document.all`.

Syntax

`<NA>`

Methods

<i>wlClick()</i> , page 326	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlTypeIn()</i> , page 366	

Properties

<i>ContainerTag</i> , page 55	<i>event</i> , page 91
<i>id</i> , page 143	<i>InnerText</i> , page 155
<i>OnClick</i> , page 190	<i>OnMouseOver</i> , page 191

See also

<i>Button</i> , page 34	<i>Checkbox</i> , page 38
-------------------------	---------------------------

<i>Collections</i> , page 47	<i>Div</i> , page 77
<i>File</i> , page 99	<i>form</i> , page 102
<i>Image</i> , page 148	<i>InputButton</i> , page 157
<i>InputCheckbox</i> , page 158	<i>InputFile</i> , page 159
<i>InputImage</i> , page 160	<i>InputRadio</i> , page 162
<i>InputText</i> , page 164	<i>Radiobutton</i> , page 210
<i>Select</i> , page 231	<i>Span</i> , page 254
<i>text</i> , page 291	<i>TextArea</i> , page 292

Url (property)

Property of Objects

<i>area</i>	<i>element</i>
<i>form</i>	<i>frames</i>
<i>Image</i>	<i>InputImage</i>
<i>link</i>	<i>location</i>
<i>window</i>	<i>wlHttp</i>

Description

Sets or retrieves the URL of the parent object on the Web page (read-only). If the parent object is of type , then this property holds the URL of the image `element`.

If the parent object is `wlGlobals`, this property holds the URL address to which the `wlGlobals` object connects.

If the parent object is `wlMeta`, then if `httpEquiv="REFRESH"` and the `content` property holds a URL, then the URL is extracted and stored in a `link` object (read-only).

Example

Area, element, form, frame, image, link, location:

```
<NA>
```

wlGlobals:

```
wlGlobals.Url = "http://www.ABCDEF.com"
```

wlMeta:

When working with `wlMeta` objects, use the all-uppercase caps form:

```
CurrentLink = document.wlMetas[0].URL
```

Comment

Notice that the URL property for `area`, `element`, `form`, `frame`, `image`, `link`, `location`, and `wlMeta` objects is written in all-uppercase caps.

This is one of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

area, page 24

content, page 56

element, page 83

frames, page 107

Image, page 148

link, page 169

Name, page 184

Url, page 302

wlBrowser, page 325

wlHttp, page 342

wlMeta, page 346

Browser configuration components, page 30

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

form, page 102

httpEquiv, page 143

InputImage, page 160

location, page 176

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

window, page 324

wlGlobals, page 339

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

UseHistory (property)

Property of Objects

wlGlobals

Description

Flag that instructs WebLOAD to use a stored set of previous commands.

Example

<NA>

HistoryLimit, page 140

wlGlobals, page 339

User-defined global properties

Property of Object

wlBrowser

Description

Global properties defined through `wlGlobals` are shared between all the threads of a single Agenda, running on a single Load Generator. Their values may be reset and changed within the main body of an Agenda. The new value will be recognized immediately by all threads sharing that global value. (Compare to the `wlSystemGlobal`, which enables sharing of global variables and values system-wide, between all threads of all Load Generators participating in a test session, and to the `wlGeneratorGlobal`, which enables sharing of global variables and values, between all threads of a single Load Generator, even when running multiple Agendas.)

Globally shared variables are useful when tracking a value or maintaining a count across multiple threads or platforms. For example, you may include these shared values in the messages sent to the Log window during a test session.

If you are working within the JavaScript code in your Agenda, you may create user-defined properties for any object. To add a user-defined property to an object, simply assign it a value. This feature is particularly useful with the `wlGlobals` object, which you can use to define your own global properties.

Use `wlGlobals` property variables to create and access variable values that you wish to share between threads of an Agenda. Edit your `wlGlobals` properties through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor*, page 18. While global variables may be accessed anywhere in your Agenda, be sure to initially declare `wlGlobals` values in the *InitAgenda()* function only. Do not define new values within the main body of an Agenda, for they will not be shared correctly by all threads.

Syntax

```
wlBrowser.UserDefinedProperty = Value
```

Example

```
function InitAgenda() {
    // create a user-defined global variable
    wlGlobals.myGlobalProperty = 10
}
```

GUI mode

Both system and user-defined global variables should be defined, set, and assigned through the Global Variables dialog boxes of the Visual AAT Properties pane. The following figure illustrates the Global UserDefined Variables dialog box with a single user-defined variable appearing.

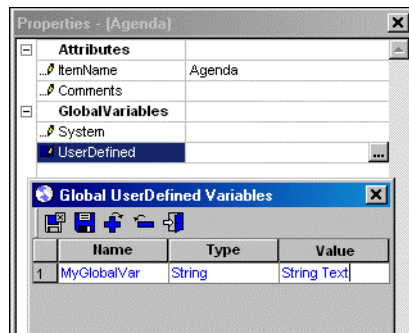


Figure 2-44: Defining a global variable

You may also work with global variables by adding or editing `wlBrowser` properties through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor*, page 18. While global variables may be accessed anywhere in your Agenda, be sure to initially declare `wlBrowser` values in the `InitAgenda()` function only. Do not define new values within the main body of an Agenda.

See also

Message functions, page 180

wlBrowser, page 325

wlGlobals, page 339

wlLocals, page 343

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGeneratorGlobal, page 334

wlHttp, page 342

wlSystemGlobal, page 363,

UserAgent (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlHttp

wlLocals

Description

Defines the type of Web browser that the Agenda should simulate, such as Navigator or Explorer. See the RadView Web site

(<http://www.radview.com/support/resourcectr/agenda.htm>) for the most up-to-date list of supported browsers.

Example

<NA>

GUI mode

WebLOAD recommends setting user agent values through the Visual AAT or Console GUI. Select a browser type and user agent through the Browser Emulation tab of the Tools | Default or Current Project Options dialog box, illustrated in the following figure:

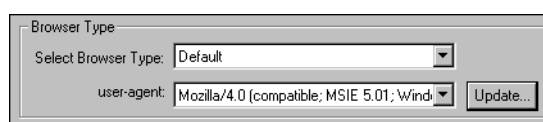


Figure 2-45: Setting user-agent value

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlLocals, page 343

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

UserName (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Properties of Objects

wlGlobals

wlHttp

wlLocals

Description

The user name that the Agenda uses to log onto a restricted HTTP site. WebLOAD automatically uses the appropriate access protocol. For example, if a site expects clients to use the NT Authentication protocol, the appropriate user name and password will be stored and sent accordingly.

GUI mode

WebLOAD by default senses the appropriate authentication configuration settings for the current test session.

If you prefer to explicitly set authentication values, WebLOAD recommends setting user authentication values through the Console GUI using one of the following approaches:

- ◆ Enter user authentication information through the Browser Dialogs tab of the Tools | Default or Current Options dialog box.
- ◆ Enter user authentication information through the Authentication for Protocol Scripts tab of the Tools | Default or Current Options dialog box. This approach is illustrated in *DefaultAuthentication*, page 66.

Syntax

You may also set user values using the `wlGlobals` properties. WebLOAD automatically sends the user name and password when a `wlHttp` object connects to an HTTP site. For example:

```
wlGlobals.UserName = "Bill"
wlGlobals.Password = "TopSecret"
```

See also

Browser configuration components, page 30

NTUserName, NTPassword, page 187

wlGlobals, page 339

wlLocals, page 343

Dialog box properties, page 73

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

UsingTimer (property)

Mode

These properties are usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The name of a timer to use for the `Get ()` or `Post ()` method.

Example

WebLOAD zeros the timer immediately before a `Get()` or `Post()` call and sends the timer value to the WebLOAD Console immediately after the call. This is equivalent to calling the `SetTimer()` and `SendTimer()` functions. Thus the following two code examples are equivalent:

```
//Version 1
wlHttp.UsingTimer = "Timer1"
wlHttp.Get("http://www.ABCDEF.com")
//Version 2
SetTimer("Timer1")
wlHttp.Get("http://www.ABCDEF.com")
SendTimer("Timer1")
```

See also

Browser configuration components, page 30

SendTimer(), page 236

SetTimer(), page 241

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Validate() (method)

Method of Object

wlBrowser

Description

Completes the specified verification tests. All failures are logged and displayed in the Log window.

Syntax

```
Validate(TestType [, ErrorLevel])
```

Parameters

TestType—One or more constants indicating type of test to be executed. If multiple tests listed, all must trigger the same type of error on failure. A complete list of verification test constants is provided in *Verification Test Components*, page 315.

ErrorLevel—Constant indicating the error level triggered if the test does not succeed. Optional. **ErrorLevel** is a very useful tool for redirecting Agenda behavior. Use the severity level to determine the execution path to be followed. Less severe errors may be noted and ignored. More severe failures may cause the whole test to be aborted. The error level may be one of the following:

- ◆ **WLSuccess**—the transaction terminated successfully.

- ◆ `WLMInorError`—this specific transaction failed, but the test session may continue as usual. The Agenda displays a warning message in the Log window and continues execution from the next statement.
- ◆ `WLError`—this specific transaction failed and the current test round was aborted. The Agenda displays an error message in the Log window and begins a new round.
- ◆ `WLSevereError`—this specific transaction failed and the test session must be stopped completely. The Agenda displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Return Value

None.

Example

```
wlBrowser.Verification.Validate(wlGlobalTests, WLMInorError)
```

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

value (property)

Property of Objects

Button

InputButton

InputRadio

element

InputCheckbox

option

*wlHeader**wlHttp.Data**wlHttp.Header**wlSearchPair***Description**

Sets and retrieves the value associated with the parent object.

When working with `elements` or `options`, this property holds the text associated with this object. This is the value that is returned to the server when a FORM control of type Button, Checkbox, Radiobutton, Reset, or Submit is submitted. Thus the `value` property holds the HTML value attribute of the object (the `<OPTION>` element). If the element does not have a value attribute, WebLOAD sets the `value` property equal to the `text` property.

When working with `wlHeader` or `wlSearchPair` objects, this property holds the value of the search key.

When working with `wlHttp.Data` or `wlHttp.Header` objects, this property holds the value of the data string being submitted through an HTTP Post command.

Syntax**For elements and options:**

```
<NA>
```

For wlHeaders:

```
document.wlHeaders[index#].value = "TextString"
```

For example:

```
document.wlHeaders[0].value = "Netscape-Enterprise/3.0F"
```

For wlSearchPairs:

```
document.links[1].wlSearchPairs[index#].value = "TextString"
```

For example:

```
document.links[1].wlSearchPairs[0].value = "OpticsResearch"
```

For wlHttp.Header:

```
wlHttp.Header["value"] = "TextString"
```

For wlHttp.Data:

When working with `wlHttp.Data` objects, use the uppercase form:

```
wlHttp.Data.Value = "SearchFor=icebergs&SearchType=ExactTerm"
```

Comment

Notice that the `Value` property for `element` and `wlHttp.Data` objects is written in uppercase.

Comment

One of the properties to which users may assign a weight for Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

<i>bitrate</i> , page 30	<i>Button</i> , page 34
<i>Checkbox</i> , page 38	<i>connectionBandwidth</i> , page 53
<i>Collections</i> , page 47	<i>currentPosition</i> , page 62
<i>currentStreamName</i> , page 62	<i>Data</i> , page 63
<i>DataFile</i> , page 65	<i>duration</i> , page 80
<i>Dynamic Object Recognition (DOR) for Dynamic HTML</i> , page 95 in the <i>WebLOAD Programming Guide</i>	<i>element</i> , page 83
<i>Erase</i> , page 87	<i>File</i> , page 99
<i>fileName</i> , page 100	<i>form</i> , page 102
<i>FormData</i> , page 104	<i>Get()</i> , page 110
<i>Header</i> , page 138	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputCheckbox</i> , page 158
<i>InputFile</i> , page 159	<i>InputImage</i> , page 160
<i>InputRadio</i> , page 162	<i>InputText</i> , page 164
<i>key</i> , page 166	<i>length</i> , page 167
<i>OpenStream()</i> , page 194	<i>option</i> , page 195
<i>Pause()</i> , page 200	<i>Play()</i> , page 200
<i>Post()</i> , page 202	<i>Radiobutton</i> , page 210
<i>Resume()</i> , page 220	<i>Select</i> , page 231
<i>state</i> , page 277	<i>Stop()</i> , page 278
<i>text</i> , page 291	<i>TextArea</i> , page 292
<i>type</i> , page 300	<i>UIContainer</i> , page 301
<i>value</i> , page 310	<i>wlClear()</i> , page 328
<i>wlHeader</i> , page 340	<i>wlHttp</i> , page 342
<i>wlMediaPlayer</i> , page 344	<i>wlMediaPlayer()</i> , page 345
<i>wlSearchPair</i> , page 356	

VCUniqueID() (function)

Description

`VCUniqueID()` provides a unique identification for the current Virtual Client instance which is unique system-wide, across multiple Load Generators, even with multiple spawned processes running simultaneously. Compare this to *ClientNum*, which provides an identification number that is only unique within a single Load Generator. The identification string is composed of a combination of the current thread number, round number, and other internal markers.

Syntax

```
VCUniqueID()
```

Parameters

None

Return Value

Returns a unique identification string for the current Virtual Client instance.

Example

```
<NA>
```

See also

ClientNum, page 43

GeneratorName(), page 108

GetOperatingSystem(), page 130

Identification variables and functions, page 146

RoundNum, page 221

VCUniqueID(), page 312

Verification (property)

Property of Object

wlBrowser

Description

When used in array form, array of items used when configuring or completing global verification tests. When used as a single property with the sub-property array `Properties[]`, stores the parameter values needed to configure or complete object-specific verification tests.

In addition to the general `Verification[]` and `Verification.Properties[]` verification access properties, each verification test includes its own list of specific properties and constants, used as parameters to customize and tailor the tests as needed. These specific verification test properties are listed in *Verification Test Components*, page 315.

Example

```
wlBrowser.Verification.Properties["TestName"] = "TextTest"
```

A more complete example of verification function use within a longer Agenda fragment appears in *Functional Testing and Reporting*, page 127 in the *WebLOAD Programming Guide*.

GUI mode

Once a test session Agenda has been recorded, WebLOAD recommends that users add verification tests to their testing Agenda or reset error levels through the convenient Visual AAT or Console GUI. Verification tests may be run either on a single Web page or on all Web pages accessed in a test session. Configure your verification test selections through the Functional Testing tab of the Tools | Default or Current Project Options dialog box, illustrated in the following figure:

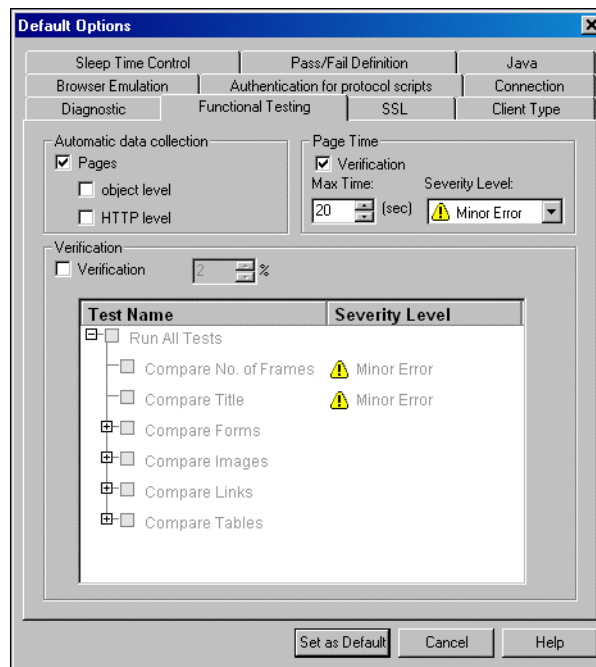


Figure 2-46: Selecting Verification Tests

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

<i>EndTransaction()</i> , page 85	<i>Functional Testing and Reporting</i> , page 127 in the <i>WebLOAD Programming Guide</i>
<i>ReportEvent()</i> , page 214	<i>SetFailureReason()</i> , page 239
<i>TimeoutSeverity</i> , page 293	<i>Transaction verification components</i> , page 297
<i>TransactionTime</i> , page 298	<i>Validate()</i> , page 309
<i>Verification Test Components</i> , page 315	<i>Verification Test Property List: Global and Page Level</i> , page 318
<i>VerificationFunction()</i> (user-defined), page 320	<i>wlBrowser</i> , page 325
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

Verification Test Components

Properties of Objects

wlBrowser

Description

Verification tests are essentially comparisons between snapshots of the DOM state stored in the expected and actual databases. Once a test session Agenda has been recorded, WebLOAD users add verification tests through the Verification Test Wizards. Use the Text, Table, and Page Verification Test Wizards to add as many verification tests as needed, covering almost every aspect of a typical client session. Test strings may be taken from input files or assigned to variables, simplifying testing with many different testing values. Tests may be defined as global defaults for all test sessions, for a complete Agenda, or for a single Web page. You can also create your own user-defined verification function, tailored to the unique aspects of the application being tested.

Global and page level tests are essentially identical. The user simply specifies through the GUI if the selected tests should be completed for every Web page in the current test session or for a single Web page only. The verification tests currently available to WebLOAD users, together with the properties associated with each test, are itemized in *Verification Test Property List: Global and Page Level*, page 318. Each test is listed by name, followed by a table that includes the WebLOAD identification constant, properties, and values associated with the test. Each verification test includes its own list of specific properties and constants, used as parameters to customize and tailor the tests as needed.

In most cases WebLOAD users select or customize their preferred verification tests through the Verification Test Wizards, described in the *WebLOAD User's Guide*, and these tests are added to the Agenda code and executed by WebLOAD automatically. The property and method descriptions and sample code fragments included in this chapter are meant only to clarify the code that you may see in the JavaScript Agendas, created for you automatically by WebLOAD. If you prefer to design a customized verification function that fills a specific testing need, add a

user-defined verification function to your Agenda as described in *Custom verification functions*, page 137 in the *WebLOAD Programming Guide*. See *VerificationFunction() (user-defined)*, page 320, for a complete syntax specification.

Syntax

Specific verification test properties are accessed through the general purpose `wlBrowser.Verification` property, which is an object in its own right. The `Verification` property links to a `Properties[]` array that provides access to the full set of properties for each verification test. Values for specific verification test properties are set through the `wlBrowser.Verification.Properties[]` array.

In addition to this general `wlBrowser` verification access property, each verification test includes its own list of specific properties and constants, used as parameters to customize and tailor the tests as needed. These specific verification test properties are listed in *Verification Test Property List: Global and Page Level*, page 318.

Example

```
wlBrowser.Verification.Properties["TestName"] = "TextTest"
```

A more complete example of verification function use within a longer Agenda fragment appears in *Functional Testing and Reporting*, page 127.

GUI mode

Once a test session Agenda has been recorded, WebLOAD recommends that users add verification tests to their testing Agenda or reset error levels through the convenient Visual AAT or Console GUI. Verification tests may be run either on a single Web page or on all Web pages accessed in a test session. Configure your verification test selections through the Functional Testing tab of the Tools | Default or Current Project Options dialog box, illustrated in the following figure:

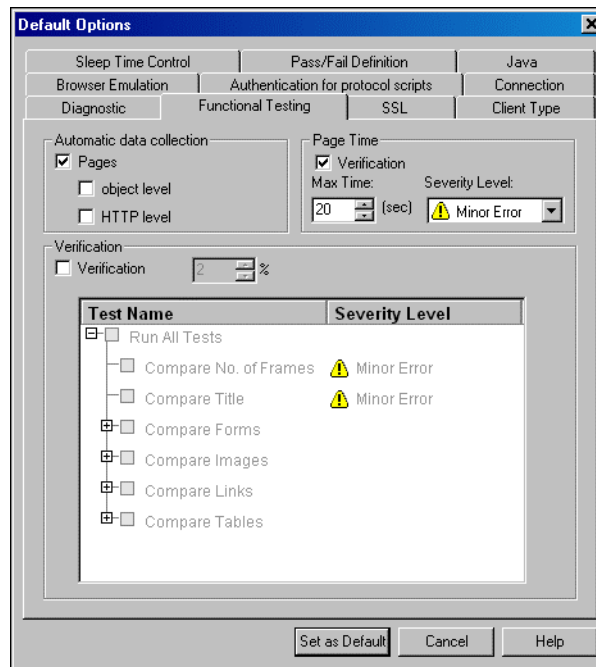


Figure 2-47: Selecting Verification Tests

Methods

Validate() (used with the *wlBrowser* object only)

Properties

Verification, page 313

In addition to this general `wlBrowser.Verification` verification access property, each verification test includes its own list of specific properties and constants, used as parameters to customize and tailor the tests as needed. These specific verification test properties are listed in the following table. Verification tests are performed on a complete Web page. Comparisons are completed between snapshots of the DOM state stored in the expected and actual databases. The constant name is the name by which WebLOAD identifies the test internally.

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

BeginTransaction(), page 28

CreateTable(), page 60

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

ReportEvent(), page 214
TimeoutSeverity, page 293
TransactionTime, page 298
Verification Test Components, page 315
VerificationFunction() (user-defined), page 320
Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*
the WebLOAD Programming Guide
SetFailureReason(), page 239
Transaction verification components, page 297
Validate(), page 309
Verification Test Property List: Global and Page Level, page 318
wlBrowser, page 325

Verification Test Property List: Global and Page Level

Test Identification Constant

Name	Description
wlGlobalTests	Run the specified global verification tests on a single Web page or on all Web pages included in a session.

Verification.Properties[] values for available tests:

Name	Description	Possible Value
wlCompareForms	Compare the following expected and actual form components: <ul style="list-style-type: none"> Number of forms Number of elements on the forms DOR match 	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
wlCompareFrames	Compare the expected and actual number of nested frames.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
wlCompareImages	Compare the number of expected and actual images and complete an optional CRC and DOR check.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
wlCompareLinks	Compare the number of expected and actual links and complete a DOR match.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
wlCompareTables	Compare the expected and actual number of tables and table structure.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
wlCompareTitle	Compare the expected and actual title.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR

WebLOAD Actions, Objects, and Functions

<code>wlLinkCheck</code>	Verify all the links on the page to make sure they are valid.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
<code>wlReportDefault Values</code>	Report current default values.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
<code>wlTextSearch</code>	Search the page for the defined expression. The test may be set to ignore white spaces or be case sensitive.	DO_NOT_CHECK REPORT_AS_MINOR_ERROR REPORT_AS_ERROR REPORT_AS_SEVERE_ERROR
<code>CRCcheck</code>	Perform a CRC confirmation check on the selected image. Optionally available with the <code>wlCompareImages</code> test.	"Yes" or "No" default value Yes
<code>TextToFind{Index}</code>	Text string for which WebLOAD should search. Included with the <code>wlTextSearch</code> test.	
<code>TextNotToFind{Index}</code>	Text string which WebLOAD should <i>not</i> find. Included with the <code>wlTextSearch</code> test.	

Global Verification Test Example

Global verification tests are completed immediately upon navigating to a new Web page, as illustrated in the following example:

```
function InitAgenda() {
wlBrowser.Verification.Properties["wlCompareImages"]=REPORT_AS_MINOR_ERROR
wlBrowser.Verification.Properties["wlTextSearch"]=REPORT_AS_SEVERE_ERROR
wlBrowser.Verification.Properties["TextToFind1"]="error"
}

// Start of Main Agenda Body
...
...
wlBrowser.ExpectNavigation("http://www.abc.com")
wlBrowser.Navigate("http://www.abc.com")
wlBrowser.SyncDOM(1)

wlBrowser.Verification.Validate(wlGlobalTests)
// Indicates that all global tests should be
// performed at this point using the GUI defaults
```

See also

Adding transactions, page 49 in the *WebLOAD Programming Guide*

CreateDOM(), page 59

BeginTransaction(), page 28

CreateTable(), page 60

Custom verification functions, page 137 in the *WebLOAD Programming Guide*

EndTransaction(), page 85

ReportEvent(), page 214

TimeoutSeverity, page 293

TransactionTime, page 298

Verification Test Components, page 315

VerificationFunction() (user-defined), page 320

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Data Drilling—WebLOAD transaction reports, page 149 in the *WebLOAD Programming Guide*

Functional Testing and Reporting, page 127 in the *WebLOAD Programming Guide*

SetFailureReason(), page 239

Transaction verification components, page 297

Validate(), page 309

Verification Test Property List: Global and Page Level, page 318

wlBrowser, page 325

VerificationFunction() (user-defined) (function)

Description

User-defined verification function to be used with a 'named' transaction. A function written by the user, tailored to the specific testing and verification needs of the application being tested.

Syntax

```
UserDefinedVerificationFunction(specified by user)
{
  ...
  <any valid JavaScript code>
  return value
}
```

Parameters

Specified by user.

Return Value

The user-defined *Verification()* function returns a value based on user-specified criterion. You define the success and failure criterion for user-defined transactions. You also determine the severity level of any failures. The severity level determines the execution path when the main Agenda resumes control. Less severe failures may be noted and ignored. More severe failures may cause the whole test to be aborted.

Set the severity level in the verification function return statement. All failures are logged and displayed in the WebLOAD Log Window, similar to any other WebLOAD test failure. Refer to Appendix A in the *WebLOAD User's Guide* for more information on return values and error codes. Transactions may be assigned one of the following return values:

- ◆ `WLSuccess`—the transaction terminated successfully.

- ◆ `WLMinorError`—this specific transaction failed, but the test session may continue as usual. The Agenda displays a warning message in the Log window and continues execution from the next statement.
- ◆ `WLError`—this specific transaction failed and the current test round was aborted. The Agenda displays an error message in the Log window and begins a new round.
- ◆ `WLSevereError`—this specific transaction failed and the test session must be stopped completely. The Agenda displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

The default return value is `WLSuccess`. If no other return value is specified for the transaction, WebLOAD will assume that the transaction terminated successfully.

Example

The following sample verification function checks if the current title of the Web page matches the page title expected at this point. (In this case, the function looks for a match with a Google page.)

```
function Transaction1_VerificationFunction()
{
    InfoMessage (document.title)
    if (document.title.indexOf ("Google") > 0)
        return WLSuccess
    else
        return WLMinorError
}
```

Comment

Note that all functions must be declared in the Agenda before they can be called.

For a more complete explanation and examples of functional testing and transaction verification, see *Functional Testing and Reporting*, page 127 in the *WebLOAD Programming Guide*.

See also

<i>Adding transactions</i> , page 49 in the <i>WebLOAD Programming Guide</i>	<i>BeginTransaction()</i> , page 28
<i>CreateDOM()</i> , page 59	<i>CreateTable()</i> , page 60
<i>Custom verification functions</i> , page 137 in the <i>WebLOAD Programming Guide</i>	<i>Data Drilling—WebLOAD transaction reports</i> , page 149 in the <i>WebLOAD Programming Guide</i>
<i>EndTransaction()</i> , page 85	<i>Functional Testing and Reporting</i> , page 127 in the <i>WebLOAD Programming Guide</i>
<i>ReportEvent()</i> , page 214	<i>SetFailureReason()</i> , page 239
<i>TimeoutSeverity</i> , page 293	<i>Transaction verification components</i> , page 297
<i>TransactionTime</i> , page 298	<i>Validate()</i> , page 309

Verification Test Components, page 315

Verification Test Property List: Global and Page Level, page 318

VerificationFunction() (user-defined), page 320

wlBrowser, page 325

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

Version (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

wlGlobals

wlHttp

wlLocals

Description

Stores the HTTP version number for the current test session. Current supported versions include 1.0 and 1.1.

Example

<NA>

GUI mode

WebLOAD recommends selecting an HTTP version through the Console GUI. Click on the appropriate version number radio button in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

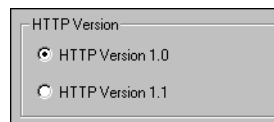


Figure 2-48: Selecting HTTP version number

See also

Browser configuration components, page 30

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wLocals, page 343*Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*

WarningMessage() (function)

Description

Use this function to display a warning message in the Log Window.

Syntax

```
WarningMessage (msg)
```

Parameters

msg—A string with a warning message to be sent to the Log window.

Return Value

None.

Example

```
<NA>
```

Comment

If you call `WarningMessage ()` in the main script, WebLOAD sends a warning message to the Log window and continues with Agenda execution as usual. The message has no impact on the continued execution of the test session.

GUI mode

WebLOAD recommends adding message functions to your Agenda files directly through the Visual AAT GUI. For example, the following figure illustrates adding a Message Node to an Agenda Tree. The corresponding code appears in the JavaScript View pane in the center and the properties are in the Properties pane on the right.

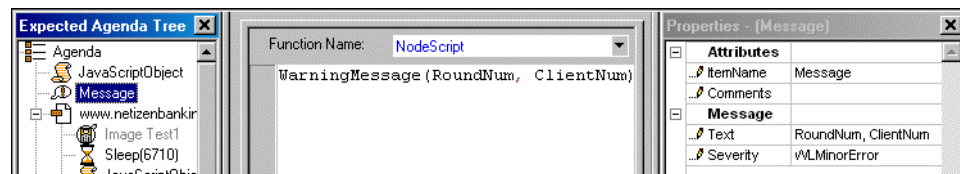


Figure 2-49: Adding a Message Node to an Agenda

See also

Error Management, page 104 in the *WebLOAD Programming Guide* *ErrorMessage()*, page 89

<i>ExceptionEnabled</i> , page 92	<i>GetMessage()</i> , page 129
<i>GetSeverity()</i> , page 133	<i>InfoMessage()</i> , page 152
<i>LiveConnect Overview</i> , page 255 in the <i>WebLOAD Programming Guide</i>	<i>Message functions</i> , page 180
<i>ReportLog()</i> , page 215	<i>SevereErrorMessage()</i> , page 245
<i>Using the IntelliSense JavaScript Editor</i> , page 18	<i>WarningMessage()</i> , page 323
<i>wlException</i> , page 331	<i>wlException()</i> object constructor, page 333

window (object)

Property of Object

frames

Description

The `window` object represents an open browser window. `window` objects store the complete parse results for downloaded HTML pages. Use the `window` object to gain access to the document in the window. From the `window` properties you can retrieve the document itself, check the location, and access other subframes that are nested within that window. Typically, the browser creates a single `window` object when it opens an HTML document. However, if a document defines one or more frames the browser creates one `window` object for the original document and one additional `window` object (a *child window*) for each frame. The child window may be affected by actions that occur in the parent. For example, closing the parent window causes all child windows to close. Note that the ‘parent’ window item is usually implicitly understood when accessing the HTML document information.

`window` objects are also accessed through nested frames, where the `frame` object’s `window` property points to a child window nested within the given frame (read-only).

Example

When working with multiple child windows of a `frames` collection, access the first child window using the following expressions:

```
frames[0]
```

or

```
document.frames[0]
```

Access the properties (document, location, or frames) of the first child window with the following expressions:

```
frames[0].<child-property>
```

or

```
document.frames[0].<child-property>
```

For example:

```
frames[0].location
```

or

```
document.frames[0].location
```

Methods

wlClose(), page 329

Properties

document, page 79

frames, page 107

location, page 176

title, page 296

Name, page 184

Url, page 302

Comment

This is one of the objects for which users may request a weighted search through Dynamic Object Recognition. See *Dynamic Object Recognition (DOR) for Dynamic HTML*, page 95 in the *WebLOAD Programming Guide*, for more information.

See also

Collections, page 47

length, page 167

wlBrowser (object)

Description

The `wlBrowser` object stores configuration information for immediate user activities, including properties defining expected dialog boxes, verification test selections, and dynamic state management. The `wlBrowser` object also contains the methods that implement the user activities saved during the Visual AAT recording session.

Properties and Methods

The `wlBrowser` object includes the following property and method classes:

Actions, page 20

Dialog box properties, page 73

Dynamic Object Recognition (DOR) components, page 81

Verification Test Components, page 315

Syntax

Each individual function class includes the syntax specifications that apply to that class.

GUI mode

The `wlBrowser` property and method descriptions explain how to explicitly set values for these session configuration properties within your JavaScript Agenda files. Note that the recommended way to set configuration values is through the WebLOAD GUI, using the Default, Current, and Global Options dialog boxes under the Tools menu in the WebLOAD desktop. The GUI dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

Actions, page 20

Dialog box properties, page 73

Dynamic Object Recognition (DOR) components, page 81

Dynamic Object Recognition (DOR) for Dynamic HTML, page 95 in the *WebLOAD Programming Guide*

ExpectNavigation(), page 97

FindObject(), page 102

Navigate(), page 186

ObjectProperty[], page 189

SetWindow(), page 244

Sleep(), page 247

Sleeping or pausing in mid-session, page 28 in the *WebLOAD Programming Guide*

SyncDOM(), page 280

Verification Test Components, page 315

wClick() (action)

Method of Objects

area

Button

Checkbox

Image

InputButton

InputCheckbox

InputImage

InputRadio

link

Radiobutton

Reset

Submit

TableCell

UIContainer

Description

Clicks the mouse button on an object, triggering an *OnClick* event.

If during the original recording session the mouse was clicked on an image, this action includes two integer parameters with the coordinates that define where within the image the mouse was clicked. The coordinates specify the offset from the upper left hand corner of the image.

If during the original recording session the mouse was clicked on a checkbox, this action includes a Boolean parameter that specifies whether or not the checkbox was actually checked at the time this click was recorded.

Syntax

```
Object.wlClick()
```

OR

```
InputImage.wlClick(offsetX, offsetY)
```

OR

```
CheckBox.wlClick(CheckStatus)
```

Parameters

offsetX—Integer value with the X coordinate of the cursor at the time this click was recorded. Applies only when clicked on an input image. Represents offset from upper left corner of image.

offsetY—Integer value with the Y coordinate of the cursor at the time this click was recorded. Applies only when clicked on an input image. Represents offset from upper left corner of image.

CheckStatus—Boolean flag indicating (*true* or *false*) whether or not the object was checked at the time this click was recorded. Applies only when clicked on a checkbox.

Return Value

None

Example

```
link1 = wIBrowser.FindObject(FT_LINK, 2)
link1.wlClick()

FlashObject1 = wIBrowser.FindObject(FT_OBJECT_ELEMENT, 0)
FlashObject1.wlClick(0.274, 0.457)
```

See also

Actions, page 20

AutoNavigate(), page 26

Button, page 34

event, page 91

Image, page 148

InputCheckbox, page 158

InputRadio, page 162

Navigate(), page 186

area, page 24

Back(), page 27

Checkbox, page 38

Forward(), page 106

InputButton, page 157

InputImage, page 160

link, page 169

OnClick, page 190

<i>OnMouseOver</i> , page 191	<i>Radiobutton</i> , page 210
<i>Refresh()</i> , page 213	<i>Reset</i> , page 218
<i>SetWindow()</i> , page 244	<i>Submit</i> , page 279
<i>TableCell</i> , page 285	<i>UIContainer</i> , page 301
<i>wlBrowser</i> , page 325	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlSubmit()</i> , page 362	<i>wlTypeIn()</i> , page 366

wClear() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects

The `wlHttp` object includes the following collections for storing data. These data storage collections each include the method `wlClear()`.

<i>wlHttp.Data</i>	<i>wlHttp.DataFile</i>
<i>wlHttp.FormData</i>	<i>wlHttp.Header</i>

Description

`wlClear()` is used to clear property values from the specified `wlHttp` data collection.

Syntax

```
wlHttp.DataCollection.wlClear([FieldName])
```

Parameters

[FieldName]—An optional user-supplied string with the name of the field to be cleared.

Return Value

None

Example

If called with no parameters, then all values set for the collection are cleared:

```

wlHttp.FormData["a"] = "DDD"
wlHttp.FormData["B"] = "FFF"
wlHttp.FormData.wlClear()
// Clear all value from all fields in FormData

InfoMessage (wlHttp.FormData["a"])
// This statement has no meaning, since there
// is currently no value assigned to "a"

If wlClear () is passed a FieldName parameter, then only the value of the specified field is
cleared:

wlHttp.FormData.wlClear ("FirstName")
// Clears only value assigned to "FirstName"

```

See also*Collections*, page 47*Data*, page 63*DataFile*, page 65*FormData*, page 104*Header*, page 138*wlHttp*, page 342*Working in HTTP Protocol Mode*, page 213 in the
WebLOAD Programming Guide

wlClose() (action)

Method of Object*window***Description**

Closes the current window.

Syntax`window.wlClose ()`**Parameters**

None

Return Value

None

Example

<NA>

See also*Actions*, page 20*AutoNavigate()*, page 26

<i>Back()</i> , page 27	<i>event</i> , page 91
<i>Forward()</i> , page 106	<i>Navigate()</i> , page 186
<i>OnClick</i> , page 190	<i>OnMouseOver</i> , page 191
<i>Refresh()</i> , page 213	<i>SetWindow()</i> , page 244
<i>window</i> , page 324	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlBrowser</i> , page 325
<i>wlMouseDown()</i> , page 347	<i>wlMouseOver()</i> , page 348
<i>wlMouseUp()</i> , page 349	<i>wlMultiSelect()</i> , page 350
<i>wlSelect()</i> , page 358	<i>wlSubmit()</i> , page 362
<i>wlTypeIn()</i> , page 366	

wlCookie (object)

Description

The `wlCookie` object sets and deletes cookies. These activities may be required by an HTTP server. Note that cookie management is usually handled automatically through the standard DOM `document.cookie` property. WebLOAD supports the `wlCookie` object as an alternate approach to cookie management. You may use the methods of `wlCookie` to create as many cookies as needed. For example, each WebLOAD client running an Agenda can set its own cookie identified by a unique name. `wlCookie` is a local object. WebLOAD automatically creates an independent `wlCookie` object for each thread of an Agenda. You cannot manually declare `wlCookie` objects yourself.

By default, WebLOAD always accepts cookies that are sent from a server. When WebLOAD connects to a server, it automatically submits any cookies in the server's domain that it has stored. The `wlCookie` object lets you supplement or override this behavior in the following ways:

- ◆ A thread can create its own cookies.
- ◆ A thread can delete cookies that it created.

Aside from these two abilities, WebLOAD does not distinguish in any way between cookies that it receives from a server and those that you create yourself. For example, if a thread creates a cookie in a particular domain, it automatically submits the cookie when it connects to any server in the domain.

Syntax

```
wlCookie.method()
```

Example

```
//Set a cookie
```

```
wlCookie.Set("CUSTOMER", "JOHN_SMITH", "www.ABCDEF.com",
            "/", "Wed, 08-Apr-98 17:29:00 GMT")
//WebLOAD submits the cookie
wlHttp.Get("www.ABCDEF.com/products/OrderForm.cgi")
//Delete the cookie
wlCookie.ClearAll()
```

Methods

ClearAll(), page 40

delete(), page 70

Set(), page 238

See also

wlException (object)

Description

Agenda scripts that encounter an error during runtime do not simply fail and die. This would not be helpful to testers who are trying to analyze when, where, and why an error in their application occurs. WebLOAD Agendas incorporate a set of error management routines to provide a robust error logging and recovery mechanism whenever possible. The `wlException` object is part of the WebLOAD error management protocol.

WebLOAD users have a variety of options for error recovery during a test session. The built-in error flags provide the simplest set of options; an informative message, a simple warning, stop the current round and skip to the beginning of the next round, or stop the test session completely. Users may also use `try()/catch()` commands to enclose logical blocks of code within a round. This provides the option of catching any minor errors that occur within the enclosed block and continuing with the next logical block of code within the current round, rather than skipping the rest of the round completely.

During a recording session, the Visual AAT automatically encloses each navigation block in the Agenda within a `try()/catch()` pair of commands. Each automatic `try()/catch()` pair delimits a single navigation block. If an error is caught while the Agenda is in the middle of executing the code within a navigation block, WebLOAD will detour to a user-defined error function and then continue execution with the next navigation block in the Agenda. Users may add their own `try()/catch()` pairs to an Agenda, delimiting their own logical code blocks and defining their own alternate set of activities to be executed in case an error occurs within that block.

`wlException` objects store information about errors that have occurred, including informative message strings and error severity levels. Users writing error recovery functions to handle the errors caught within a `try()/catch()` pair may utilize the `wlException`

object. Use the `wLException` methods to perhaps send error messages to the Log Window or trigger a system error of the specified severity level.

Example

The following code fragment illustrates a typical error-handling routine:

```

InitAgenda() {
    wLBrowser.ExceptionEnabled=True
}
try{
    ...
    //do a lot of things
    ...
    wLBrowser.Navigate("http://www.abc.com")
    //error occurs here
    ...
}
catch(e) {
    //things to do in case of error
    if (e.GetSeverity() == WLError) {
        // Do one set of Error activities
        e.ReportLog()
    }
    else {
        // Do a different set of Severe Error activities
        throw e
    }
}
}

```

Note that `wLBrowser.ExceptionEnabled` must be set to `True` in the `InitAgenda()` function to be able to use the `wLException` object later in the Agenda. WebLOAD by default always sets this property to `True` automatically.

Methods

GetMessage(), page 129

GetSeverity(), page 133

ReportLog(), page 215

wLException() object constructor, page 333

Comment

`wLGlobals.ExceptionEnabled` must be set to `True` in the `InitAgenda()` function to be able to use the `wLException` object later in the Agenda. WebLOAD by default always sets this property to `True`. This is illustrated in the preceding example.

See also

Error Management, page 104 in the *WebLOAD Programming Guide*

ErrorMessage(), page 89

ExceptionEnabled, page 92

InfoMessage(), page 152

LiveConnect Overview, page 255 in the
WebLOAD Programming Guide

SevereErrorMessage(), page 245

WarningMessage(), page 323

Message functions, page 180

Using the IntelliSense JavaScript Editor, page 18

wlException, page 331

wlException() (constructor)

Method of Object

wlException

Description

Creates a new `wlException` object.

Syntax

```
NewExceptionObject = new wlException(severity, message)
```

Parameters

severity—One of the following integer constants:

- ◆ `WLError`—this specific transaction failed and the current test round was aborted. The Agenda displays an error message in the Log window and begins a new round.
- ◆ `WLSevereError`—this specific transaction failed and the test session must be stopped completely. The Agenda displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

message—The exception message stored as a text string.

Return Value

Returns a new `wlException` object.

Example

```
myUserException=new wlException(WLError, "Invalid date")
```

See also

Error Management, page 104 in the *WebLOAD Programming Guide*

ExceptionEnabled, page 92

GetSeverity(), page 133

LiveConnect Overview, page 255 in the
WebLOAD Programming Guide

ReportLog(), page 215

ErrorMessage(), page 89

GetMessage(), page 129

InfoMessage(), page 152

Message functions, page 180

SevereErrorMessage(), page 245

Using the IntelliSense JavaScript Editor, page 18

WarningMessage(), page 323

wlException, page 331

wlException() object constructor, page 333

wGeneratorGlobal (object)

Description

WebLOAD provides a global object called `wGeneratorGlobal`. The `wGeneratorGlobal` object enables sharing of global variables and values between all threads of a single Load Generator, even when running multiple Agendas. (Compare to the `wSystemGlobal`, which enables sharing of global variables and values system-wide, between all threads of all Load Generators participating in a test session, and to the `wGlobals`, which enables sharing of global variables and values between threads of a single Agenda, running on a single Load Generator.)

Globally shared variables are useful when tracking a value or maintaining a count across multiple threads or platforms. For example, you may include these shared values in the messages sent to the Log window during a test session.

WebLOAD creates exactly one `wGeneratorGlobal` object for each Load Generator participating in a test session. Use the `wGeneratorGlobal` methods to create and access variable values that you wish to share between threads of a Load Generator. Edit `wGeneratorGlobal` properties and methods through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor*, page 18. While global variables may be accessed anywhere in your Agenda, be sure to initially declare `wGeneratorGlobal` values in the *InitAgenda () function only*. Do not define new values within the main body of an Agenda, for they will not be shared correctly by all threads.

Syntax

<NA>

Methods

The `wGeneratorGlobal` object includes the following methods:

Add(), page 21

Get(), page 110

Set(), page 237

Properties

`wGeneratorGlobal` incorporates a dynamic property set that consists of whatever global variables have been defined, set, and accessed by the user through the `wGeneratorGlobal` method set only.

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGet() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

Each of the different types of collections of elements found in the parsed DOM tree include the method `wlGet()`.

Description

`wlGet()` is used when getting data from a property in the collection to distinguish between keywords and user-defined variables that share the same names. The need for this care is explained in this section.

Syntax

```
Collection.wlGet(PropertyName)
```

Parameters

PropertyName—A string with the name of the property whose value is to be gotten.

Return Value

The value of the specified property

Example

```
document.forms[0].elements.wlGet("FirstName")
```

Comment

In JavaScript, users may work interchangeably with either an `array[index]` or `array.index` notation. For example, the following two references are interchangeable:

```
wlHttp.FormData["Sunday"]
```

or

```
wlHttp.FormData.Sunday
```

This flexibility is convenient for programmers, who are able to select the syntax that is most appropriate for the context. However, it could potentially lead to ambiguity. For example, assume a Web site included a form with a field called `length`. This could lead to a confusing situation, where the word `length` appearing in an Agenda could represent either the number

of elements in a `FormData` array, as explained in *length*, page 167, or the value of the `length` field in the form. Errors would arise from a reasonable assignment statement such as:

```
wlHttp.FormData["length"] = 7
```

This is equivalent to the illegal assignment statement:

```
wlHttp.FormData.length = 7
```

WebLOAD therefore uses `wIGet()` to retrieve field data whenever the name could lead to potential ambiguity. When recording Agendas with the Visual AAT, WebLOAD recognizes potential ambiguities and inserts the appropriate `wIGet()` statements automatically.

See also

Collections, page 47

wlHttp, page 342

wIGetAllForms() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

document

Description

Retrieve a collection of all forms (<FORM> elements) in an HTML page and its nested frames.

Syntax

```
wIGetAllForms()
```

Parameters

None

Return Value

A collection that includes the forms in the top-level frame (from which you called the method) and all its subframes at any level of nesting.

Example

```
<NA>
```

See also

Browser configuration components, page 30

document, page 79

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

wlGetAllFrames() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object

document

Description

Retrieve a collection of all frames in an HTML page, at any level of nesting.

Syntax

```
wlGetAllFrames ()
```

Parameters

None

Return Value

A collection that includes the top-level frame (from which you called the method) and all its subframes.

Example

<NA>

See also

Browser configuration components, page 30 *document*, page 79

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

wlGetAllLinks() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Object*document***Description**

Retrieve a collection of all links (<A> elements) in an HTML page and its nested frames.

Syntax

```
wlGetAllLinks ()
```

Parameters

None

Return Value

A collection that includes links in the top-level frame (from which you called the method) and all its subframes at any level of nesting.

Example

```
<NA>
```

See also

Browser configuration components, page 30 *document*, page 79

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

wIGlobals (object)

Description

The `wIGlobals` object stores the default global configuration properties set by the user through the Visual AAT or Console GUI, including properties defining expected dialog boxes, verification test selections, and dynamic state management.

`wIGlobals` is a global object, whose property values are shared by all threads of an Agenda running on a single Load Generator. The `wIGlobals` object enables sharing of user-defined global variables and values between threads of a single Agenda, running on a single Load Generator. (Compare to the `wlGeneratorGlobal`, which enables sharing of global variables and values between all threads of a single Load Generator, and the `wlSystemGlobal`, which enables sharing of global variables and values system-wide, between all threads of all Load Generators participating in a test session.)

Note that most global configuration property values and user-defined variables should be set through the Visual AAT or Console GUI, as described in the *WebLOAD User's Guide*. The property descriptions here are intended mainly to explain the lines of code seen in the JavaScript View of the Visual AAT desktop. Syntax details are also provided for the benefit of users who prefer to manually edit the JavaScript code of their Agendas through the IntelliSense

editor, described in *Using the IntelliSense JavaScript Editor*, page 18. If you do decide to edit the global variable values in your Agenda, set `wlGlobals` properties in the `InitAgenda()` function only. Do not define new values within the main body of an Agenda. The values will not be shared correctly by all Agenda threads.

The configuration properties of the `wlGlobals` object are almost all duplicated in the `wlLocals`, which contains the local configuration settings for browser actions, and in the `wlHttp`, which contains configuration settings that are limited to a single specific browser action. To understand how there could potentially be three different settings for a single configuration property, see *Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide*.

Properties

The `wlGlobals` object includes the following property classes:

Automatic State Management for HTTP Protocol Mode, page 25 *Browser configuration components*, page 30

Transaction verification components, page 297 *User-defined global properties*, page 304

Syntax

Each individual property class includes the syntax specifications that apply to that class.

GUI mode

The `wlGlobals` property and method descriptions explain how to explicitly set values for these session configuration properties within your JavaScript Agenda files. Note that the recommended way to set configuration values is through the WebLOAD GUI, using the Default, Current, and Global Options dialog boxes under the Tools menu in the Visual AAT or Console desktop. The GUI dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide* *wlGeneratorGlobal*, page 334

wlHttp, page 342

wlLocals, page 343

wlSystemGlobal, page 363

wlHeader (object)

Mode

The `wlHeader` object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

When working in the HTTP Protocol mode, headers on a Web page are accessed through `wlHeader` objects that are grouped into collections of `wlHeaders`. The `wlHeaders` collection is a property of the following objects:

document

Description

Each `wlHeader` object contains a key-value pair. `wlHeader` objects provide access to the key/value pairs in the HTTP *response headers*. (Information found in *request headers* is available through the `wlHttp.Header` property. For key-value pairs found in *URL search strings*, see *wlSearchPair*.)

`wlHeader` objects are local to a single thread. You cannot create new `wlHeader` objects using the JavaScript `new` operator, but you can access them through the properties and methods of the standard DOM objects. `wlHeader` properties are read only.

Syntax

`wlHeader` objects are grouped together within collections of `wlHeaders`. To access an individual `wlHeader`'s properties, check the `length` property of the `wlHeaders` collection and use an index number to access the individual `wlHeader` object, with the following syntax:

```
NumberOfHeaderObjects = document.wlHeaders.length
document.wlHeaders[index#].<wlHeader-property>
```

Example

WebLOAD stores the header pairs from the most recent Get, Post, or Head command in the `document.wlHeaders` collection. The following statement would retrieve an HTTP header:

```
wlHttp.Head("http://www.ABCDEF.com")
```

For a header that looks something like this:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/3.0F
Date: Sun, 11 Jan 1998 08:25:20 GMT
Content-type: text/html
Connection: close
Host: Server2.MyCompany.com
```

WebLOAD parses the header pairs as follows:

```
document.wlHeaders[0].key = "Server"
document.wlHeaders[0].value = "Netscape-Enterprise/3.0F"
document.wlHeaders[1].key = "Date"
document.wlHeaders[1].value = "Sun, 11 Jan 1998 08:25:20 GMT"
...
```

Properties

The `wlHeader` object includes the following properties:

key, page 166

value, page 310

See also

Collections, page 47

document, page 79

Header, page 138

wlSearchPair, page 356

wlHtml (object)

Mode

The `wlHtml` object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Description

If your Agenda downloads HTML code, you can use the `wlHtml` object to retrieve parsed elements of the code. The `wlHtml` object also lets you retrieve the HTTP header fields and status and parse URL addresses into their host, port, and URI components.

`wlHtml` is a local object. WebLOAD automatically creates an independent `wlHtml` object for each thread of an Agenda. You cannot manually declare `wlHtml` objects yourself.

Syntax

<NA>

Methods

GetFieldValue(), page 115

GetFieldValueInForm(), page 116

GetFormAction(), page 117

GetFrameByUrl(), page 118

GetFrameUrl(), page 119

GetHeaderValue(), page 120

GetHost(), page 121

GetHostName(), page 122

GetLinkByName(), page 127

GetLinkByUrl(), page 128

GetPortNum(), page 131

GetQSFieldValue(), page 132

GetStatusLine(), page 134

GetStatusNumber(), page 135

GetUri(), page 136

See also

wHttp (object)

Description

The `wHttp` object stores configuration information for immediate user activities, including properties defining expected dialog boxes, verification test selections, and dynamic state management. Many of these properties are duplicated in the `wGlobals`, which contains the default global configuration settings for browser actions, and in the `wLocals`, which contains the local configuration settings for browser actions. To understand how there could potentially be three different settings for a single configuration property, see *Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide*. The `wHttp` object also contains the methods that implement the user activities saved during the Visual AAT recording session. User activities may be recreated through one of two approaches: the high-level User Action mode or the low-level HTTP Protocol mode. Methods for each of these testing modes are included in the `wHttp` object.

Properties and Methods

The `wHttp` object includes the following property and method classes:

Automatic State Management for HTTP Protocol Mode, page 25

Transaction verification components, page 297

Syntax

Each individual function class includes the syntax specifications that apply to that class.

GUI mode

The `wHttp` property and method descriptions explain how to explicitly set values for these session configuration properties within your JavaScript Agenda files. Note that the recommended way to set configuration values is through the WebLOAD GUI, using the Default, Current, and Global Options dialog boxes under the Tools menu in the Visual AAT or Console desktop. The GUI dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wGlobals, page 339

wLocals, page 343

wlLocals (object)

Description

The `wlLocals` object stores the local default configuration information for user activities, such as the URL, user name and password, proxy server, and dialog box management. `wlLocals` is a local object. WebLOAD creates an independent `wlLocals` object for *each thread* of an Agenda. You cannot declare `wlLocals` objects yourself.

The properties of the `wlLocals` object are all duplicated in the `wlGlobals`, which contains the default global settings, and in the `wlHttp`, which contains the settings for an immediate action. To understand how there could potentially be three different settings for a single configuration property, see *Rules of scope for local and global variables*, page 111 in the *WebLOAD Programming Guide*.

Properties

The `wlLocals` object includes the following property classes:

Automatic State Management for HTTP Protocol *Browser configuration components*, page 30
Mode, page 25

Transaction verification components, page 297

Syntax

Each individual function class includes the syntax specifications that apply to that class.

GUI mode

The `wlLocals` property and method descriptions explain how to explicitly set values for these session configuration properties within your JavaScript Agenda files. Note that the recommended way to set configuration values is through the WebLOAD GUI, using the Default, Current, and Global Options dialog boxes under the Tools menu in the Visual AAT or Console desktop. The GUI dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

wlGlobals, page 339

wlHttp, page 342

wlMediaPlayer (object)

Description

WebLOAD provides support for streaming media applications through the `wlMediaPlayer` object. Each `wlMediaPlayer` object handles streaming media data, such as an audio or video clip. `wlMediaPlayer` objects are local to a single thread.

Syntax

`wlMediaPlayerObject.Property`
`wlMediaPlayerObject.Method()`

Example

Each individual property includes examples of the syntax for that property.

Methods

<i>OpenStream()</i> , page 194	<i>Pause()</i> , page 200
<i>Play()</i> , page 200	<i>Resume()</i> , page 220
<i>Stop()</i> , page 278	<i>wlMediaPlayer()</i> object constructor, page 345

Properties

<i>bitrate</i> , page 30	<i>connectionBandwidth</i> , page 53
<i>currentPosition</i> , page 62	<i>currentStreamName</i> , page 62
<i>duration</i> , page 80	<i>fileName</i> , page 100
<i>state</i> , page 277	<i>type</i> , page 300

See also

wlMediaPlayer() (constructor)

Method of Object

wlMediaPlayer

Description

Returns a new `wlMediaPlayer` object.

Syntax

`new wlMediaPlayer()`

Parameters

None

Return Value

A pointer to a new `wlMediaPlayer` object.

Example

```
MyMediaPlayerObject = new wlMediaPlayer()
```

See also

bitrate, page 30

currentPosition, page 62

duration, page 80

OpenStream(), page 194

Play(), page 200

state, page 277

type, page 300

wlMediaPlayer() object constructor, page 345

connectionBandwidth, page 53

currentStreamName, page 62

fileName, page 100

Pause(), page 200

Resume(), page 220

Stop(), page 278

wlMediaPlayer, page 344

wlMeta (object)

Mode

The `wlMeta` object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

When working in the HTTP Protocol mode, META objects on a Web page are accessed through `wlMeta` objects that are grouped into collections of `wlMetas`. The `wlMetas` collection is a property of the following objects:

document

Description

Each `wlMeta` object stores the parsed data for an HTML meta object (<META> tag). `wlMeta` objects are local to a single thread. You cannot create new `wlMeta` objects using the JavaScript `new` operator, but you can access them through the properties and methods of the standard DOM objects. `wlMeta` properties are read only.

Syntax

`wlMeta` objects are grouped together within collections of `wlMetas`. To access an individual `wlMeta`'s properties, check the `length` property of the `wlMetas` collection and use an index number to access the individual `wlMeta` object, with the following syntax:

```
NumberOfMetaObjects = document.wlMetas.length
document.wlMetas[#].<wlMeta-property>
```

Example

To find out how many `wlMeta` objects are contained within a document header, check the value of:

```
document.wlMetas.length
```

Access each `wlMeta`'s properties directly using the preceding syntax. For example:

```
document.wlMetas[0].key
```

Properties

The `wlMeta` object includes the following properties:

content, page 56

httpEquiv, page 143

Name, page 184

Url, page 302

See also

Collections, page 47

document, page 79

length, page 167

wMouseDown() (action)

Method of Objects

area

Button

Image

InputButton

InputImage

link

TableCell

UIContainer

Description

Presses down on the mouse button over an object.

Syntax

```
Object.wMouseDown()
```

Parameters

None

Return Value

None

Example

```
link1 = wlBrowser.FindObject(FT_LINK, 2)
link1.wlMouseDown()
```

See also

<i>Actions</i> , page 20	<i>area</i> , page 24
<i>AutoNavigate()</i> , page 26	<i>Back()</i> , page 27
<i>Button</i> , page 34	<i>event</i> , page 91
<i>Forward()</i> , page 106	<i>Image</i> , page 148
<i>InputButton</i> , page 157	<i>InputImage</i> , page 160
<i>link</i> , page 169	<i>Navigate()</i> , page 186
<i>OnClick</i> , page 190	<i>OnMouseOver</i> , page 191
<i>Refresh()</i> , page 213	<i>SetWindow()</i> , page 244
<i>TableCell</i> , page 285	<i>UIContainer</i> , page 301
<i>wlBrowser</i> , page 325	<i>wlClick()</i> , page 326
<i>wlClose()</i> , page 329	<i>wlMouseDown()</i> , page 347
<i>wlMouseOver()</i> , page 348	<i>wlMouseUp()</i> , page 349
<i>wlMultiSelect()</i> , page 350	<i>wlSelect()</i> , page 358
<i>wlSubmit()</i> , page 362	<i>wlTypeIn()</i> , page 366

wlMouseOver() (action)

Method of Objects

<i>area</i>	<i>Button</i>
<i>Image</i>	<i>InputButton</i>
<i>InputImage</i>	<i>link</i>
<i>TableCell</i>	<i>UIContainer</i>

Description

Rolls the mouse over an object.

Syntax

```
Object.wlMouseOver()
```

Parameters

None

Return Value

None

Example

```
link1 = wlBrowser.FindObject (FT_LINK, 2)
link1.wlMouseOver ()
```

See also*Actions*, page 20*AutoNavigate()*, page 26*Button*, page 34*Forward()*, page 106*InputButton*, page 157*link*, page 169*OnClick*, page 190*Refresh()*, page 213*TableCell*, page 285*wlBrowser*, page 325*wlClose()*, page 329*wlMouseOver()*, page 348*wlMultiSelect()*, page 350*wlSubmit()*, page 362*area*, page 24*Back()*, page 27*event*, page 91*Image*, page 148*InputImage*, page 160*Navigate()*, page 186*OnMouseOver*, page 191*SetWindow()*, page 244*UIContainer*, page 301*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseUp()*, page 349*wlSelect()*, page 358*wlTypeIn()*, page 366

wlMouseUp() (action)

Method of Objects*area**Image**InputImage**TableCell**Button**InputButton**link**UIContainer***Description**

Releases the mouse button over an object.

Syntax*Object*.wlMouseUp ()

Parameters

None

Return Value

None

Example

```
link1 = wlBrowser.FindObject(FT_LINK, 2)
...
link1.wlMouseUp()
```

See also*Actions*, page 20*AutoNavigate()*, page 26*Button*, page 34*Forward()*, page 106*InputButton*, page 157*link*, page 169*OnClick*, page 190*Refresh()*, page 213*TableCell*, page 285*wlBrowser*, page 325*wlClose()*, page 329*wlMouseOver()*, page 348*wlMultiSelect()*, page 350*wlSubmit()*, page 362*area*, page 24*Back()*, page 27*event*, page 91*Image*, page 148*InputImage*, page 160*Navigate()*, page 186*OnMouseOver*, page 191*SetWindow()*, page 244*UIContainer*, page 301*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseUp()*, page 349*wlSelect()*, page 358*wlTypeIn()*, page 366

wlMultiSelect() (action)

Method of Object*Select***Description**

Selects one or more objects from a list on the Web page while the control key is pressed. Pointers to each selected object are stored in the method parameters.

Syntax

```
Object.wlMultiSelect("Option1", "Option2", ...)
```

Parameters

Option1 . . . n—Pointers to the one or more objects being selected.

Return Value

None

Example

<NA>

See also

Actions, page 20

Back(), page 27

Forward(), page 106

OnClick, page 190

Refresh(), page 213

Select, page 231

wlClose(), page 329

wlMouseDown(), page 347

wlMouseUp(), page 349

wlSelect(), page 358

wlTypeIn(), page 366

AutoNavigate(), page 26

event, page 91

Navigate(), page 186

OnMouseOver, page 191

SetWindow(), page 244

wlClick(), page 326

wlBrowser, page 325

wlMouseOver(), page 348

wlMultiSelect(), page 350

wlSubmit(), page 362

wLObjectFile (object)

Description

The wLObjectFile object writes Agenda output messages to a global output file. Create wLObjectFile objects and manage your files using the constructor and methods described in this section.

Syntax

```
MyFileObj = new wLObjectFile("filename")
...
MyFileObj.Write("Happy Birthday")
...
delete MyFileObj
```

Example

Each individual property includes examples of the syntax for that property.

GUI mode

The `wlOutputFile` objects and methods described in this chapter may be added directly to the code in a JavaScript Object within an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Users who are programming their own JavaScript Object code within their Agenda may still take advantage of the Visual AAT GUI to simplify their programming efforts. Rather than manually type out the code to create a `wlOutputFile` object, with the risk of making a mistake, even a trivial typo, and adding invalid code to the Agenda file, users may select the Edit | Insert | JavaScript | General menu to bring up a list of available `wlOutputFile` constructor and methods, as illustrated in the following figure. The Visual AAT automatically inserts the correct code for the selected command into the JavaScript Object currently being edited. The user may then change the parameters without any concerns about mistakes in the object syntax.

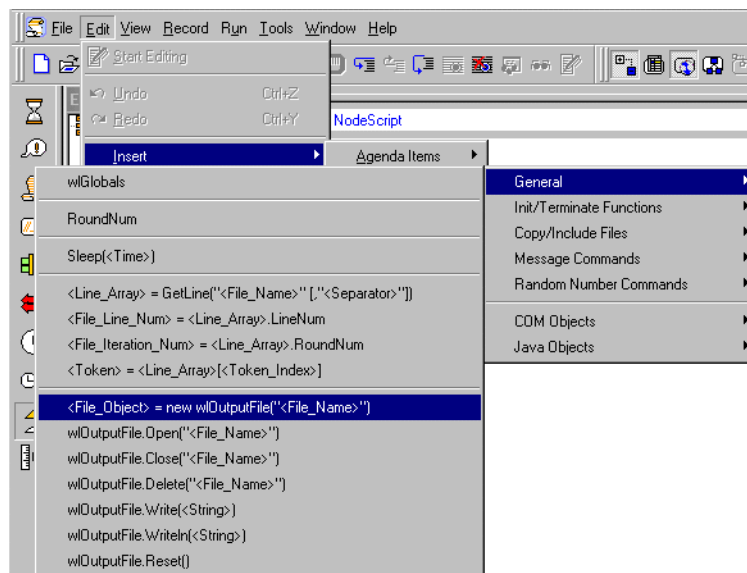


Figure 2-50: `wlOutputFile` object methods list

Methods

`wlOutputFile()` object constructor, page 354

`Close()`, page 45

`Open()`, page 193

`Write()`, page 372

`delete()`, page 70

`Reset()`, page 219

`WriteLn()`, page 373

Comment

You may also use the WebLOAD functions listed here to open and close output files.

- ◆ To **open** an output file:

```
Open (filename)
```

- ◆ To **close** an output file:

```
Close (filename)
```

When you use the `Close ()` function to close a file data will be flashed to the disk.

The `wlOutputFile` object saves Agenda output messages. To save server response data, use the `wlBrowser.Outfile` property.

Note

Declaring a new `wlOutputFile` object creates a new, empty output file. If a file of that name already exists, the file will be completely overwritten. Information will not be appended to the end of an existing file. Be sure to choose a *unique filename* for the new output file if you do not want to overwrite previous Agenda data.

If you declare a new `wlOutputFile` object in the `InitAgenda ()` function of an Agenda, the output file will be shared by all the Agenda threads. There is no way to specify a specific thread writing sequence—each thread will write to the output file in real time as it reaches that line in the Agenda execution.

If you declare a new `wlOutputFile` object in the `InitClient ()` function or main body of an Agenda, use the thread number variable as part of the new filename to be sure that each thread will create a unique output file.

If you declare a new `wlOutputFile` object in the main body of an Agenda, and then run your Agenda for multiple iterations, use the `RoundNum` variable as part of the new filename to be sure that each new round will create a unique output file.

Generally, you should only create new `wlOutputFile` objects in the `InitAgenda ()` or `InitClient ()` functions of an Agenda, not in the main script. If a statement in the main script creates an object, *a new object is created each time the statement is executed*. If WebLOAD repeats the main script many times, a large number of objects may be created and the system may run out of memory.

See also

CopyFile(), page 57

GetLine(), page 124

Using the Form Data Wizard, page 35 in the *WebLOAD Programming Guide*

File management functions, page 100

IncludeFile(), page 150

Using the IntelliSense JavaScript Editor, page 18

wlOutputFile() (constructor)

Method of Object

wlOutputFile

Description

To create a new `wlOutputFile` object, use the `wlOutputFile()` constructor.

Syntax

```
new wlOutputFile(filename)
```

Parameters

`filename`—Name of the new output file to be created.

Return Value

A pointer to a new `wlOutputFile` object.

Example

```
MyFileObj = new wlOutputFile("FileName")
```

Note

Declaring a new `wlOutputFile` object creates a new, empty output file. If a file of that name already exists, the file will be completely overwritten. Information will not be appended to the end of an existing file. Be sure to choose a *unique filename* for the new output file if you do not want to overwrite previous Agenda data.

If you declare a new `wlOutputFile` object in the `InitAgenda()` function of an Agenda, the output file will be shared by all the Agenda threads. There is no way to specify a specific thread writing sequence—each thread will write to the output file in real time as it reaches that line in the Agenda execution.

If you declare a new `wlOutputFile` object in the `InitClient()` function or main body of an Agenda, use the thread number variable as part of the new filename to be sure that each thread will create a unique output file.

If you declare a new `wlOutputFile` object in the main body of an Agenda, and then run your Agenda for multiple iterations, use the `RoundNum` variable as part of the new filename to be sure that each new round will create a unique output file.

Ideally, create new `wlOutputFile` objects only in the `InitAgenda()` function of an Agenda, not in the main script. If a statement in the main script creates an object, a new object is created *each time the statement is executed*. If WebLOAD repeats the main script many times, a large number of objects may be created and the system may run out of memory.

See also

<i>Close()</i> , page 45	<i>CopyFile()</i> , page 57
<i>delete()</i> , page 70	<i>File management functions</i> , page 100
<i>GetLine()</i> , page 124	<i>IncludeFile()</i> , page 150
<i>Open()</i> , page 193	<i>Reset()</i> , page 219
<i>Using the Form Data Wizard</i> , page 35 in the <i>WebLOAD Programming Guide</i>	<i>Using the IntelliSense JavaScript Editor</i> , page 18
<i>wlOutputFile()</i> , page 351	<i>wlOutputFile()</i> , page 354
<i>Write()</i> , page 372	<i>WriteLn()</i> , page 373

wlRand (object)

Description

The wlRand object is a random number generator.

wlRand is a local object. WebLOAD automatically creates an independent wlRand object for the test session Agenda. You cannot declare wlRand objects yourself.

Syntax

```
wlRand.Method()
```

Example

The following example generates three random numbers having the following possible values:

- ◆ Any integer.
- ◆ An integer from 1 to 9.
- ◆ One of the three numbers 0, 1, or 1.5.

```
function InitAgenda() {
    wlRand.Seed(23)
}

AnyInteger = wlRand.Num()
OneToNine = wlRand.Range(1, 9)
OneOfThreeNumbers = wlRand.Select(0, 1, 1.5)
```

GUI mode

wlRand object command lines may be added directly to the code in a JavaScript Object within an Agenda through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor*, page 18. Users who are programming their own JavaScript Object code within their Agenda may still take advantage of the Visual AAT GUI to simplify their programming efforts. Rather than manually type out the code for a random number function, with the risk of making

a mistake, even a trivial typo, and adding invalid code to the Agenda file, users may click on the Random Number Commands command line in the Edit | Insert | JavaScript | General menu and select a `wlRand` method to add to their code. The Visual AAT automatically inserts the correct code for the random number function into the JavaScript Object currently being edited. The user may then change the parameters without any concerns about mistakes in the function syntax.

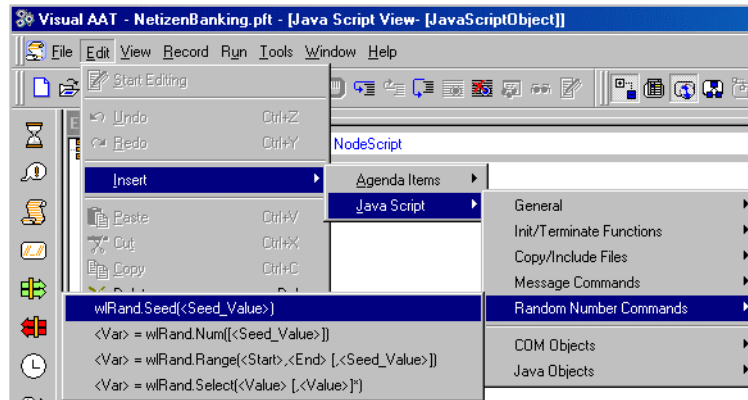


Figure 2-51: wlRand commands

Methods

Num(), page 188

Range(), page 211

Seed(), page 231

Select(), page 232

See also

wlSearchPair (object)

Method of Object

link

location

Description

Each `wlSearchPair` object contains a parsed version of the search attribute string, storing the key/value pairs found in a document's *URL search strings*. (For key-value pairs found in HTTP *response headers*, see `wlHeader`. Information found in *request headers* is available through the `wlHttp.Header` property.)

wSearchPair objects are grouped into wSearchPairs collections, where the collections are themselves properties of the link and location objects.

wSearchPair objects are local to a single thread. You cannot create new wSearchPair objects using the JavaScript new operator, but you can access them through the properties and methods of the standard DOM objects. wSearchPair properties are read only.

Syntax

wSearchPair objects are grouped together within collections of wSearchPairs. To access an individual wSearchPair's properties, check the length property of the wSearchPairs collection and use an index number to access the individual wSearchPair object, with the following syntax:

```
NumberOfSearchPairObjects =
    document.links[1].wSearchPairs.length
document.links[1].wSearchPairs[index#].<wSearchPair-property>
```

Example

To find out how many wSearchPair objects are contained within a document's link, check the value of:

```
document.links[1].wSearchPairs.length
```

Access each wSearchPair's properties directly through the index number of that item. For example:

```
document.links[1].wSearchPairs[0].key
```

Suppose that the third link on a Web page has the following HTML code:

```
<A href="http://www.ABCDEF.com/ProductFind.exe?
    Product=modems&Type=ISDN"> ISDN Modems </A>
```

You can download the page and parse the links using the following Agenda:

```
function InitAgenda() {
    wGlobals.Url = "http://www.ABCDEF.com"
    //Enable link parsing
    wGlobals.ParseLinks = "Yes"
}
wLHttp.Get()
```

For the link in question, WebLOAD stores the attribute pairs in the document.links[2].wSearchPairs property. This property is actually a collection containing two wSearchPair objects. The following is a complete listing of the collection.

```
document.links[2].wSearchPairs[0].key = "Product"
document.links[2].wSearchPairs[0].value = "modems"
document.links[2].wSearchPairs[1].key = "Type"
document.links[2].wSearchPairs[1].value = "ISDN"
```

Properties

The `wlSearchPair` object includes the following properties:

key, page 166

value, page 310

See also

Collections, page 47

Header, page 138

link, page 169

location, page 176

wlHeader, page 340

wlHttp, page 342

wlSelect() (action)

Method of Object

Select

Description

Selects a single object from a list on the Web page. A pointer to the selected object is stored in the method parameter.

Syntax

Object.`wlSelect`("Option")

Parameters

Option—A string, placed in quotation marks, that contains the name of the option selected from a list.

Return Value

None

Example

<NA>

See also

Actions, page 20

AutoNavigate(), page 26

Back(), page 27

event, page 91

Forward(), page 106

Navigate(), page 186

OnClick, page 190

OnMouseOver, page 191

Refresh(), page 213

Select, page 231

SetWindow(), page 244*w1Click()*, page 326*w1MouseDown()*, page 347*w1MouseUp()*, page 349*w1Select()*, page 358*w1TypeIn()*, page 366*w1Browser*, page 325*w1Close()*, page 329*w1MouseOver()*, page 348*w1MultiSelect()*, page 350*w1Submit()*, page 362

w1Set() (method)

Mode

This method is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Method of Objects

The `w1Http` object includes the following collections for storing data. These data storage collections each include the method `w1Set()`.

*w1Http.Data**w1Http.DataFile**w1Http.FormData**w1Http.Header*

Description

`w1Set()` is used when assigning a value to an element in the collection, to distinguish between keywords and user-defined variables that share the same names. The need for this care is explained in this section.

Syntax

```
w1Http.Collection.w1Set(FieldName, Value)
```

Parameters

FieldName—A string with the name of the field whose value is to be set.

Value—The value to be assigned to the specified field.

Return Value

The value of the specified property.

Example

```
w1Http.FormData.w1Set("FirstName", "Bill")
```

Comment

In JavaScript, users may work interchangeably with either an `array[index]` or `array.index` notation. For example, the following two references are interchangeable:

```
wlHttp.FormData["Sunday"]
```

or

```
wlHttp.FormData.Sunday
```

This flexibility is convenient for programmers, who are able to select the syntax that is most appropriate for the context. However, it could potentially lead to ambiguity. For example, assume a Web site included a form with a field called `length`. This could lead to a confusing situation, where the word `length` appearing in an Agenda could represent either the number of elements in a `FormData` array, as explained in *length*, page 167, or the value of the `length` field in the form. Errors would arise from a reasonable assignment statement such as:

```
wlHttp.FormData["length"] = 7
```

This is equivalent to the illegal assignment statement:

```
wlHttp.FormData.length = 7
```

WebLOAD therefore uses `wlSet()` to set field data whenever the name could lead to potential ambiguity. When recording Agendas with the AAT, WebLOAD recognizes potential ambiguities and inserts the appropriate `wlSet()` statements automatically. In this case:

```
wlHttp.FormData.wlSet("length", 7)
```

See also

Collections, page 47

Data, page 63

DataFile, page 65

FormData, page 104

Header, page 138

wlHttp, page 342

Working in HTTP Protocol Mode, page 213 in the *WebLOAD Programming Guide*

wlSource (property)

Mode

The `wlSource` property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

document

Description

The complete HTML source code of the frame (read-only string).

You can use the HTML source to search for any desired information in an HTML page. For information on JavaScript searching capabilities, see *Regular Expressions* in the *Netscape JavaScript Guide*, which is supplied with the WebLOAD software.

Syntax

```
document.wlSource
```

Comment

To use the HTML source when working in HTTP Protocol mode, you must enable the *SaveSource* property of the *wlGlobals*, *wlLocals*, or *wlHttp* object. To save the source in a file, use the *Outfile* property.

See also

document, page 79

SaveSource, page 226

Outfile, page 197

wlStatusLine (property)

Mode

The *wlStatusLine* property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

document

Description

The status line of the HTTP header (read-only string, for example "OK").

Syntax

```
document.wlStatusLine
```

See also

document, page 79

wlStatusNumber (property)

Mode

The `wlStatusNumber` property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

document

Description

The HTTP status value, which WebLOAD retrieves from the HTTP header (read-only integer, for example 200).

Syntax

```
document.wlStatusNumber
```

See also

document, page 79

wlSubmit() (action)

Method of Object

form

Description

Submits the form pointed to by the current `form` object. Recorded when the user presses the ENTER key to submit a form.

Syntax

```
form.wlSubmit()
```

Parameters

None

Return Value

None

Example

<NA>

See also*Actions*, page 20*Back()*, page 27*form*, page 102*Navigate()*, page 186*OnMouseOver*, page 191*SetWindow()*, page 244*wlClick()*, page 326*wlMouseDown()*, page 347*wlMouseUp()*, page 349*wlSelect()*, page 358*wlTypeIn()*, page 366*AutoNavigate()*, page 26*event*, page 91*Forward()*, page 106*OnClick*, page 190*Refresh()*, page 213*wlBrowser*, page 325*wlClose()*, page 329*wlMouseOver()*, page 348*wlMultiSelect()*, page 350*wlSubmit()*, page 362

wlSystemGlobal (object)

Description

WebLOAD provides a global object called `wlSystemGlobal`. The `wlSystemGlobal` object enables sharing of global variables and values between all elements of a test session, across multiple Agendas running on multiple Load Generators. (Compare to the `wlGeneratorGlobal`, which enables sharing of global variables and values between all threads of a single Load Generator, and to the `wlGlobals`, which enables sharing of global variables and values between all threads of a single Agenda, running on a single Load Generator.)

Globally shared variables are useful when tracking a value or maintaining a count across multiple threads or platforms. For example, you may include these shared values in the messages sent to the Log window during a test session.

WebLOAD creates exactly one `wlSystemGlobal` object per a test session. Use the `wlSystemGlobal` object methods to create and access variable values that you wish to share system-wide. Edit `wlSystemGlobal` object properties and methods through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor*, page 18. While global variables may be accessed anywhere in your Agenda, be sure to initially declare `wlSystemGlobal` values in the *InitAgenda () function only*. Do not define new values within the main body of an Agenda, for they will not be shared correctly by all threads.

Syntax

<NA>

Methods*Add()*, page 21*Get()*, page 110*Set()*, page 237**Properties**

`wlSystemGlobal` incorporates a dynamic property set that consists of whatever global variables have been defined, set, and accessed by the user through the `wlSystemGlobal` method set only.

See also

Rules of scope for local and global variables, page 111 in the *WebLOAD Programming Guide*

User-defined global properties, page 304

wlGeneratorGlobal, page 334

wlTable (object)

Mode

The `wlTable` object is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

When working in the HTTP Protocol mode, TABLE objects on a Web page are accessed through `wlTable` objects that are grouped into collections of `wlTables`. The `wlTables` collection is a property of the following object:

document

Description

Each `wlTable` object contains the parsed data for an HTML table (<TABLE> tag), and serves as a means of providing access to the cells of the HTML table. Because table data is organized into rows and cells, the `wlTable` object is also linked to `row` and `cell` objects and their properties.

`wlTable` objects are grouped together within collections of `wlTables`. The tables are arranged in the order in which they appear on the HTML page.

Syntax

To access an individual `wlTable`'s properties, check the `length` property of the `wlTables` collection and use an index number to access the individual `wlTable` object, with the following syntax:

```
NumberOfTableObjects = document.wlTables.length
```



```
document.wlTables[index#].<wTable-property>
```

Example

Access each wTable's properties directly through the index number of that item. For example:

```
document.wlTables[0].cols
```

wTable objects may also be accessed directly using the table ID. This is illustrated in the *id* property description.

Properties

Each wTable object contains information about the data found in the whole table, organized by rows, columns, and cells. The wlTables object includes the following properties:

<i>cell</i> , page 35 (wTable and row property)	<i>cols</i> , page 48 (wTable property)
<i>id</i> , page 143 (wTable property)	<i>row</i> , page 223 (wTable property)

Comment

WebLOAD recommends managing tables on a Web page through the standard `document.all` collection. For example, rather than using the following approach to access the first table on a page:

```
myFirstTableObject = document.wlTables[0]
```

WebLOAD recommends accessing the first table on a page through the following:

```
myFirstTableObject = document.all.tags("TABLE")[0]
```

See also

<i>cellIndex</i> , page 37 (cell property)	<i>Collections</i> , page 47
<i>Compare()</i> , page 49	<i>CompareColumns</i> , page 51
<i>CompareRows</i> , page 52	<i>Details</i> , page 72
<i>document</i> , page 79	<i>InnerHTML</i> , page 153 (cell property)
<i>InnerText</i> , page 155 (cell property)	<i>MatchBy</i> , page 178
<i>Prepare()</i> , page 205	<i>ReportUnexpectedRows</i> , page 216
<i>rowIndex</i> , page 225 (row property)	<i>TableCompare</i> , page 286
<i>tagName</i> , page 289 (cell property)	<i>wTable</i> , page 364
<i>Working in HTTP Protocol Mode</i> , page 213 in the <i>WebLOAD Programming Guide</i>	

wlTarget (property)

Mode

This property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Object

wlHttp

Description

The exact location within the Web page of the frame into which the transaction should be downloaded.

Syntax

```
wlHttp.wlTarget = "LocationString"
```

Comment

`wlTarget` uses the WebLOAD shorthand notation, described in *Identifying frame locations*, page 224 in the *WebLOAD Programming Guide*. For example, assume the expected location is set to #1.#1. Since frame numbering begins with 0, this refers to the second subframe located within the second frame on the Web page. Neither frame has been assigned an optional name value.

The `wlHttp.wlTarget` property of a transaction stores the complete path of the frame, from the root window of the Web page. Compare this to the `form.target` and `link.target` properties, which identify the most recent, immediate location of the target frame using the name string or keyword that was assigned to that frame. The last field of the `wlHttp.wlTarget` string is the target name stored in the `form.target` and `link.target` properties.

See also

Identifying frame locations, page 224 in the
WebLOAD Programming Guide

wlHttp, page 342

wlTypeIn() (action)

Method of Objects

File

InputFile

InputPassword

InputText

text

TextArea

Description

Type the new text, letter by letter, into the text area of the object in focus, recreating the text input activity recorded by the user.

Syntax

```
Object.wTypeIn("Text")
```

Parameters

Text—Text typed in by the user while the selected object was in focus.

Return Value

None

Example

The following code fragment submits a text string as a query to a Web page search engine:

```
...
Form1 = wBrowser.FindObject(FT_FORM, 4, "Query")
Text1 = Form1.FindObject(Input_Text, 1, "searchstring")
Text1.wTypeIn("WebLOAD")
Submit1 = Form1.FindObject(FT_INPUT_SUBMIT, 2, "search")
Submit1.wClick()
...
```

See also

Actions, page 20

Back(), page 27

File, page 99

InputFile, page 159

InputText, page 164

OnClick, page 190

Refresh(), page 213

text, page 291

wBrowser, page 325

wClose(), page 329

wMouseOver(), page 348

wMultiSelect(), page 350

wSubmit(), page 362

AutoNavigate(), page 26

event, page 91

Forward(), page 106

InputPassword, page 161

Navigate(), page 186

OnMouseOver, page 191

SetWindow(), page 244

TextArea, page 292

wClick(), page 326

wMouseDown(), page 347

wMouseUp(), page 349

wSelect(), page 358

wTypeIn(), page 366

wlVersion (property)

Mode

The `wlVersion` property is usually inserted manually only when working in the HTTP Protocol mode. See *Working in HTTP Protocol Mode*, page 213 in the *WebLOAD Programming Guide*, for more information.

Property of Objects

document

Description

The HTTP protocol version, which WebLOAD retrieves from the HTTP header (read-only string, for example "1.1").

Example

```
currentVersionNumber = document.wlVersion
```

GUI mode

WebLOAD recommends setting the HTTP version through the Console GUI. Click on the appropriate radio button in the Browser Emulation tab of the Tools | Default Options dialog box, illustrated in the following figure:

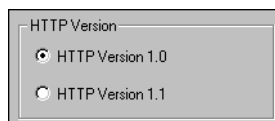


Figure 2-52: Selecting HTTP Version 1.0

See also

document, page 79

WLXmlDocument() (constructor)

Method of Object

wlXmIs

Description

Call `WLXmlDocument()` without any parameters to create a new, blank XML DOM object. The new object may be filled later with any new data you prefer. If the DTD section of your XML document includes any external references, use this form of the `WLXmlDocument()`

constructor to create new XML DOM objects. You may add nodes and post the new XML data to a Web site as described in *Working with the XML DOM*, page 161 in the *WebLOAD Programming Guide*.

Call `WLXmlDocument()` with a string parameter to create new XML DOM objects from an XML string that includes a completely self-contained DTD section with no external references.

Syntax

```
new WLXmlDocument([xmlString])
```

Parameters

[xmlString]—Optional string parameter that contains a complete set of XML document data.

Return Value

Returns a new XML DOM object. If the constructor was called with no parameters, the new object will be empty. If the constructor was called with an XML string, the new object will contain an XML DOM hierarchy based on the XML data found in the parameter string.

Example

```
NewBlankXMLObj = new WLXmlDocument()
or
NewXMLObj = new WLXmlDocument(xmlStr)
```

Comment

Objects created by the `WLXmlDocument()` constructor provide access to the XML DOM Document Interface. They do not expose the HTML property set, (`id`, `innerHTML`, and `src`), as those properties have no meaning for XML DOM objects created this way.

See also

<i>Collections</i> , page 47	<i>id</i> , page 143
<i>InnerHTML</i> , page 153	<i>load()</i> , page 170
<i>load() and loadXML() method comparison</i> , page 172	<i>loadXML()</i> , page 175
<i>src</i> , page 255	<i>wXmIs</i> , page 370
<i>XMLDocument</i> , page 380	

wXmIs (object)

Property of Object

document

Description

WebLOAD has extended the standard IE Browser DOM `document` object with the `wlXmls` collection of XML DOM objects, providing full access to XML structures. Using XML DOM objects, WebLOAD Agendas are able to both *access* XML site information, and *generate new XML data* to send back to the server for processing, taking advantage of all the additional benefits that XML provides.

Both WebLOAD and the IE Browser use the MSXML parser to create XML DOM objects. Since WebLOAD XML DOM objects and Browser XML DOM objects are created by the same MSXML parser, the XML DOM objects that are produced for both WebLOAD and the IE Browser are identical.

When working through the IE Browser, XML DOM objects are found in the `all` collection. When working through WebLOAD, XML DOM objects are found in the `wlXmls` collection. Since a WebLOAD XML DOM object is identical to an IE Browser XML DOM object, the WebLOAD XML DOM uses the same Document Interface (programming methods and properties) found in the IE Browser XML DOM.

This section describes the `wlXmls` collection and the properties and methods used most often when working with WebLOAD XML DOM objects. For an explanation of the XML DOM, see *Working with the XML DOM*, page 161 in the *WebLOAD Programming Guide*. For a complete list of the XML DOM properties and methods supported by WebLOAD, see Appendix B, *WebLOAD-supported XML DOM Interfaces*, page 391.

Syntax

XML DOM objects are grouped together within `wlXmls` collections. The XML DOM objects are arranged in the order in which they appear on the HTML page.

To access an individual XML DOM object's data and Document Interface, check the `length` property of the `wlXmls` collection and use an index number to access the individual XML DOM object.

Access the *HTML properties* for each XML DOM object directly using the following syntax:

```
document.wlXmls[#].<html-DOM property>
```

Access the *XML DOM Document Interface* for each document element directly using the following syntax:

```
document.wlXmls[#].XMLDocument.documentElement.<property>
```

Example

To find out how many XML DOM objects are contained within a document, check the value of:

```
document.wlXmls.length
```

Access the HTML property `src` as follows:

```
document.wlXmls[0].src
```

Access the XML DOM document interface as follows:

```
document.wIXmls[0].XMLDocument.documentElement.nodeName
```

XML DOM objects may also be accessed directly using the XML ID. For example, if the first XML object on a page is assigned the ID tag `myXmlDoc`, you could access the object using any of the following:

```
MyBookstore = document.wIXmls[0]
```

or

```
MyBookstore = document.wIXmls.myXmlDoc
```

or

```
MyBookstore = document.wIXmls["myXmlDoc"]
```

The following example illustrates HTML property usage. Assume you are working with a Web Bookstore site that includes the following inventory database code fragment:

```
<xml ID="xmlBookSite">
<?xml version="1.0"?>
<!-- Bookstore inventory database -->
<bookstore>
  <book>
    <author>Mark Twain</author>
    <title>Tom Sawyer</title>
    <price>$11.00</price>
  </book>
  <book>
    <author>Oscar Wilde</author>
    <title>The Giant And His Garden</title>
    <price>$8.00</price>
  </book>
</bookstore>
</xml>
```

When accessing this Web site, your Agenda may use the standard HTML properties `id` and `innerHTML` to print out text strings showing the information found within the XML tags, as follows:

```
var XMLBookstoreDoc = document.wIXmls[0]
InfoMessage("ID = " + XMLBookstoreDoc.id)
InfoMessage("HTML text = " + XMLBookstoreDoc.innerHTML)
```

Running this Agenda produces the following output:

```
ID = xmlBookSite
HTML text = <?xml version="1.0"?>
...etc.
```

Methods and Properties

WebLOAD supports all standard W3C XML DOM properties and methods, listed in Appendix B, *WebLOAD-supported XML DOM Interfaces*, page 391. These HTML properties and

methods are accessed via the `XMLDocument` property. In addition, if the object is constructed from a Data Island, the `id`, `InnerHTML`, and `src` HTML properties are exposed. Each property is described in its own section.

id, page 143

InnerHTML, page 153

load(), page 170

loadXML(), page 175

src, page 255

WLXmlDocument(), page 368

XMLDocument, page 380

Comment

WebLOAD recommends managing XML DOM objects on a Web page through the standard `document.all` collection rather than using the `wlXmLs` family of objects.

See also

Collections, page 47

document, page 79

load() and loadXML() method comparison, page 172

Write() (method)

Method of Object

wlOutputFile

Description

This method writes a string to the output file.

Syntax

```
Write(string)
```

Parameters

string—The text string you wish added to the output.

Return Value

None.

Example

```
MyFileObj = new wlOutputFile(filename)
...
MyFileObj.Write("Happy Birthday")
```

See also

Close(), page 45

CopyFile(), page 57

<i>delete()</i> , page 70	<i>File management functions</i> , page 100
<i>GetLine()</i> , page 124	<i>IncludeFile()</i> , page 150
<i>Open()</i> , page 193	<i>Reset()</i> , page 219
<i>Using the Form Data Wizard</i> , page 35 in the <i>WebLOAD Programming Guide</i>	<i>Using the IntelliSense JavaScript Editor</i> , page 18
<i>wlOutputFile</i> , page 351	<i>wlOutputFile()</i> , page 354
<i>Write()</i> , page 372	<i>Writeln()</i> , page 373

Writeln() (method)

Method of Object

wlOutputFile

Description

This method writes a string followed by a newline character to the output file.

Syntax

```
Writeln(string)
```

Parameters

string—The text string you wish added to the output.

Return Value

None.

Example

```
MyFileObj = new wlOutputFile(filename)
...
MyFileObj.Writeln("Happy Birthday")
```

See also

<i>Close()</i> , page 45	<i>CopyFile()</i> , page 57
<i>delete()</i> , page 70	<i>File management functions</i> , page 100
<i>GetLine()</i> , page 124	<i>IncludeFile()</i> , page 150
<i>Open()</i> , page 193	<i>Reset()</i> , page 219
<i>Using the Form Data Wizard</i> , page 35 in the <i>WebLOAD Programming Guide</i>	<i>Using the IntelliSense JavaScript Editor</i> , page 18
<i>wlOutputFile</i> , page 351	<i>wlOutputFile()</i> , page 354
<i>Write()</i> , page 372	<i>Writeln()</i> , page 373

WSComplexObject (object)

Description

The `WSComplexObject` object is part of the WebLOAD WebServices test suite. If the Web Services tool being accessed during a test session involves complex object types, it may be necessary to manually edit the Agenda code, as described in *Using the IntelliSense JavaScript Editor*, page 18. WebLOAD provides the `WSComplexObject` object to handle complex object types.

Example

The following code fragment illustrates work with a complex Web Services object:

```
function InitClient() {
    var Factory = new WSWebService("Factory",
        "http://qa39-ntsp6:8080/axis/services/factory?wsdl");
    Worker = new WSComplexObject();
    Worker.setType("Worker");
    Worker.addProperty("Name", "string");
    Worker.addProperty("Age", "integer");
}

Worker.setValue("Name", "Robert");
Worker.setValue("Age", 28);
var retVal = factory.addWorker( Worker );
InfoMessage( retVal );
```

Methods

addProperty(), page 22

setType(), page 242

setValue(), page 243

WSComplexObject(), page 375

Proprietary methods specific to each Web Service tool, as listed in the WSDL files

Comment

The new complex object defined in the preceding example is based on information found in the public WSDL file associated with the Web Service tool being tested. In the preceding example, the following methods were used to create and work with the complex object needed by the Web Services `Factory` tool:

- ◆ The `setType()` method defines the *type* of the new object being created. In this example, we are creating a `Worker` object for the `Factory` Web Service tool.
- ◆ The `addProperty()` method adds properties to the new `Worker` object that correspond to the properties defined for this object in the WSDL file. In this case, each `Worker` object includes two properties: `Name` and `Age`.

- ◆ The `setValue()` method assigns values to the newly added properties. In this case, the `Worker` object is assigned values for `Name`, (`Robert`), and `Age`, (`28`).
- ◆ The `addWorker()` method that appears in the preceding example is not a `WebLOAD` method. `addWorker()` is proprietary to the `Factory` Web Service tool. Use of this method is based on information provided by the Web Service's WSDL file. See *Working with Web Services*, page 30 in the *WebLOAD Programming Guide*, for more information.

Note that new `WSComplexObject` objects must be created, and all their properties added, in the `InitClient()` function. Otherwise, a new object will be created with each iteration of the `Agenda` during a test session and the system will quickly run out of memory. This is illustrated in the preceding example. Values may be assigned to the existing properties, and complex object may be accessed and used, at any point in the `Agenda` file.

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

WSGetSimpleValue(), page 376

WSWebService(), page 379

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

WSWebService, page 377

WSComplexObject() (constructor)

Method of Object

WSComplexObject

Description

Creates a new `WSComplexObject` object.

Syntax

```
newComplexObject = new WSComplexObject();
```

Parameters

None.

Return Value

Returns a new `WSComplexObject` object.

Example

```
function InitClient() {
    var Factory = new WSWebService("Factory",
        "http://qa39-ntsp6:8080/axis/services/factory?wsdl");
    Worker = new WSComplexObject();
    Worker.setType("Worker");
}
```

```

Worker.addProperty("Name", "string");
Worker.addProperty("Age", "integer");
}

```

Comment

New `WSComplexObject` objects must be created, and their properties defined, within the `InitClient()` function. Otherwise, a new object will be created with each iteration of the Agenda during a test session and the system will quickly run out of memory. This is illustrated in the preceding example.

See also

Working with Web Services, page 30, in the
WebLOAD Programming Guide

Working with Web Services complex types, page
207, in the *WebLOAD Programming Guide*

`addProperty()`, page 22

`setType()`, page 242

`setValue()`, page 243

`WSComplexObject`, page 374

`WSGetSimpleValue()`, page 376

`WSWebService`, page 377

`WSWebService()`, page 379

WSGetSimpleValue() (function)

Description

Extracts a simple value that has been stored within an XML string structure.

Syntax

```
var rVal = WSGetSimpleValue(retVal);
```

Parameters

`retVal`—The XML string structure returned by a Web Services function call.

Return Value

The simple value, (integer, string, Boolean, etc.), stored within an XML structure.

Example

```

var BNQuoteService = new WSWebService("BNQuoteService",
    "http://www.xmethods.net/sd/2001/BNQuoteService.wsdl");

// Call the getPrice() method, assign the XML return value
// structure to retVal
var retVal = BNQuoteService.getPrice("12344");

// Convert the return value structure to the expected
// simple numerical value
var rVal = WSGetSimpleValue(retVal);

```

```
// Verify that the actual value matches the expected value
if (rVal != 5.9)
    InfoMessage("getPrice() method call failed");
else
    InfoMessage("getPrice() method call succeeded");
```

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

addProperty(), page 22

setType(), page 242

WSComplexObject, page 374

WSWebService, page 377

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

SaveWSTransaction, page 228

setValue(), page 243

WSComplexObject(), page 375

WSWebService(), page 379

WSWebService (object)

Description

Many programs today take advantage of Web Services, publicly available tools that simplify the creation and maintenance of complex, large scale applications. As a complete testing platform, WebLOAD provides full testing support for the Web Services tools included in the application being tested. The WebServices Wizard, described in *Working with Web Services*, page 30, in the *WebLOAD Programming Guide*, simplifies the addition of Web Services testing to your Agenda. The WebServices Wizard handles most Web Services testing scenarios. For the occasional Web Services tools that involve complex object types, testers may have to work with the code within the testing Agenda, as described in *Working with Web Services complex types*, page 207, in the *WebLOAD Programming Guide*. This *Reference Manual* documents the syntax of the objects and methods used to implement the WebLOAD WebServices test suite.

The `WSWebService` object is part of the WebLOAD WebServices test suite. Use the WebServices Wizard to define a new `WSWebService` object for each Web Services tool being tested. Each `WSWebService` object stores information based on the WSDL file associated with a Web Service tool. In most cases, you select the methods of the Web Services tool that you wish tested through the WebServices Wizard. The corresponding lines of code are added automatically to your Agenda. When dealing with complex object types, you may have to add some lines of code within the Agenda file.

Example

The following code fragment illustrates a typical Web Service object:

```
function InitClient() {
    var BNQuoteService = new WSWebService("BNQuoteService",
        "http://www.xmethods.net/sd/2001/BNQuoteService.wsdl");
}
```

```
var retVal = BNQuoteService.getPrice("12344");
```

Methods

WSWebService(), page 379

Proprietary methods specific to each Web Services tool, as listed in the corresponding WSDL file

Comment

New `WSWebService` objects must be created in the `InitClient()` function. Otherwise, a new object will be created with each iteration of the Agenda during a test session and the system will quickly run out of memory. This is illustrated in the preceding example.

The `getPrice()` method that appears in the preceding example is not a WebLOAD method. `getPrice()` is proprietary to the `BNQuoteService` Web Services tool. The WebServices Wizard added the line that calls this method to the Agenda file when the test session designer selected this method for testing.

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

addProperty(), page 22

setValue(), page 243

WSComplexObject(), page 375

WSWebService(), page 379

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

setType(), page 242

WSComplexObject, page 374

WSGetSimpleValue(), page 376

WSWebService() (constructor)

Method of Object

WSWebService

Description

Creates a new `WSWebService` object.

Syntax

```
var WebServiceToolName = new
    WSWebService("WebServiceToolName", "WSDLurl")
```

Parameters

`WebServiceToolName`—Name of the Web Service tool being accessed, a text string.

`WSDLurl`—Location of the WSDL file associated with this Web Service tool, a text string.

Return Value

Returns a new `WSWebService` object.

Example

```
function InitClient() {
    var BNQuoteService = new WSWebService("BNQuoteService",
        "http://www.xmethods.net/sd/2001/BNQuoteService.wsdl");
}
```

Comment

New `WSWebService` objects must be created in the `InitClient()` function. Otherwise, a new object will be created with each iteration of the Agenda during a test session and the system will quickly run out of memory. This is illustrated in the preceding example.

See also

Working with Web Services, page 30, in the *WebLOAD Programming Guide*

addProperty(), page 22

setValue(), page 243

WSComplexObject(), page 375

WSWebService, page 377

Working with Web Services complex types, page 207, in the *WebLOAD Programming Guide*

setType(), page 242

WSComplexObject, page 374

WSGetSimpleValue(), page 376

XMLDocument (property)

Method of Object

`wlXmIs`

Description

The `XMLDocument` property represents the actual XML DOM object. Through `XMLDocument` you are able to access all the standard XML DOM properties and methods listed in Appendix B, *WebLOAD-supported XML DOM Interfaces*, page 391.

Syntax

Use the following syntax:

```
document.wlXmIs[#].XMLDocument.documentElement.<property>
```

`XMLDocument` is also understood by default. You may access the XML DOM properties and methods without including `XMLDocument` in the object reference. For example:

```
document.wlXmIs[0].documentElement.<property>
```

However, including `XMLDocument` is a good programming practice, to emphasize the fact that you are dealing directly with an XML DOM object and not a Data Island.

Example

```
document.wlXmIs[0].XMLDocument.documentElement.nodeName
```

Comment

WebLOAD recommends managing XML DOM objects on a Web page through the standard `document.all` collection rather than using the `wlXmIs` family of objects.

See also

Collections, page 47

InnerHTML, page 153

loadXML(), page 175

src, page 255

id, page 143

load(), page 170

load() and loadXML() method comparison, page 172

wlXmIs, page 370

WebLOAD-supported SSL Protocol Versions

SSL handshake combinations

WebLOAD supports a variety of SSL versions, ranging from the earlier SSL versions and up to the most current TLS versions. The following table illustrates the results of different handshake combinations, depending on the Client and Server SSL version:

Table A-1: SSL handshake combinations

Client setting	Server Setting			
	Undetermined	3.0W/2.0Hello	3.0 Only	2.0 Only
Undetermined	3.0	3.0	(a)	2.0
3.0W/2.0Hello	3.0	3.0	(a)	(b)
3.0 Only	3.0	3.0	3.0	(c)
2.0 Only	3.0	3.0	3.0	2.0

Each entry specifies the negotiated protocol version. In the noted instances, negotiation is impossible for the following reasons:

- (a) These protocols all support SSL 3.0, but the SSL 3.0 Only setting on the server prevents the SSL 2.0 Hello message sent by the client from being recognized.
- (b) The SSL 2.0 Hello message sent by the client is recognized, but the SSL 2.0 Only setting on the server sends a 2.0 response. The client rejects this response as it is set to communicate using only SSL 3.0.

- (c) The SSL 3.0 Hello message sent by the client will not be understood by the SSL 2.0 only server.

Commercial browsers and servers generally act as if they are set for **SSL_Version_Undetermined**, unless SSL 2.0 is disabled, in which case they act as if they are set for **SSL_Version_3_0_With_2_0_Hello**.

SSL protocols—complete list

WebLOAD supports the following SSL protocols. The protocols are sorted by bit limit. A short description follows each item, including noting whether or not the protocol is cleared for export.

128-bit encryption

The following SSL protocols work with 128-bit encryption.

- ◆ **SSL_RSA_WITH_3DES_EDE_CBC_MD5**
RSA key exchange with MD5 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
- ◆ **SSL_RSA_WITH_IDEA_CBC_MD5**
RSA key exchange with MD5 hashing and IDEA encryption in CBC mode. No export.
- ◆ **TLS_DHE_DSS_WITH_RC4_128_SHA**
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
- ◆ **TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA**
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
- ◆ **TLS_ECDH_anon_WITH_RC4_128_SHA**
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
- ◆ **TLS_ECMQV_ECNRV_WITH_3DES_EDE_CBC_SHA**
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
- ◆ **TLS_ECMQV_ECNRV_WITH_RC4_128_SHA**
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
- ◆ **TLS_ECMQV_ECDSA_WITH_3DES_EDE_CBC_SHA**

- Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve DSA key exchange with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
- ◆ TLS_ECMQV_ECDSA_WITH_RC4_128_SHA
Elliptic Curve Menezes-Qu-Vanstone key exchange with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
 - ◆ TLS_ECDH_ECNRN_WITH_3DES_EDE_CBC_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
 - ◆ TLS_ECDH_ECNRN_WITH_RC4_128_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
 - ◆ TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
 - ◆ TLS_ECDH_ECDSA_WITH_RC4_128_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
 - ◆ TLS_ECDHE_ECNRN_WITH_3DES_EDE_CBC_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
 - ◆ TLS_ECDHE_ECNRN_WITH_RC4_128_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
 - ◆ TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
 - ◆ TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
 - ◆ TLS_ECES_ECNRN_WITH_3DES_EDE_CBC_SHA
Elliptic Curve encryption scheme and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
 - ◆ TLS_ECES_ECNRN_WITH_RC4_128_SHA
Elliptic Curve encryption scheme and Elliptic Curve NRA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.

- ◆ TLS_ECES_ECDSA_WITH_3DES_EDE_CBC_SHA
Elliptic Curve encryption scheme and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit triple DES encryption using EDE in CBC mode. No export.
- ◆ TLS_ECES_ECDSA_WITH_RC4_128_SHA
Elliptic Curve encryption scheme and Elliptic Curve DSA analog with SHA-1 hashing and 128-bit ARC4/RC4 encryption. No export.
- ◆ TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
Anonymous Diffie-Hellman key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_DH_anon_WITH_RC4_128_MD5
Anonymous Diffie-Hellman key exchange with MD5 hashing and 128-bit ARC4/RC4 encryption. No export.
- ◆ TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
Ephemeral Diffie-Hellman with RSA key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
Diffie-Hellman with RSA key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
Diffie-Hellman key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_RSA_WITH_3DES_EDE_CBC_SHA
RSA key exchange with SHA-1 hashing and triple DES 128-bit encryption using EDE in CBC mode. No export.
- ◆ TLS_RSA_WITH_IDEA_CBC_SHA
RSA key exchange with SHA-1 hashing and IDEA encryption in CBC mode. For export.
- ◆ TLS_RSA_WITH_RC4_128_SHA
RSA key exchange with SHA-1 hashing and ARC4/RC4 128-bit encryption. No export.
- ◆ TLS_RSA_WITH_RC4_128_MD5
RSA key exchange with MD5 hashing and ARC4/RC4 128-bit encryption. No export.

56-bit encryption

The following SSL protocols work with 56-bit encryption.

- ◆ **TLS_ECDH_ECDSA_EXPORT_WITH_RC4_56_SHA**
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 56-bit ARC4/RC4 encryption. For export.
- ◆ **SSL_RSA_WITH_DES_CBC_MD5**
RSA key exchange with MD5 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ **TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA**
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and 56-bit ARC4/RC4 encryption. For export.
- ◆ **TLS_RSA_EXPORT1024_WITH_RC4_56_SHA**
RSA key exchange with SHA-1 hashing and 56-bit ARC4/RC4 encryption. For export.
- ◆ **TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA**
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and 56-bit DES encryption in CBC mode. For export.
- ◆ **TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA**
RSA key exchange with SHA-1 hashing and 56-bit DES encryption in CBC mode. For export.
- ◆ **TLS_ECDH_anon_WITH_DES_CBC_SHA**
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ **TLS_ECMQV_ECNRV_WITH_DES_CBC_SHA**
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve NRA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ **TLS_ECMQV_ECDSA_WITH_DES_CBC_SHA**
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve DSA key exchange with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ **TLS_ECDH_ECNRV_WITH_DES_CBC_SHA**
Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ **TLS_ECDH_ECDSA_WITH_DES_CBC_SHA**
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.

- ◆ TLS_ECDHE_ECNRA_WITH_DES_CBC_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ TLS_ECDHE_ECDSA_WITH_DES_CBC_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ TLS_ECES_ECNRA_WITH_DES_CBC_SHA
Elliptic Curve encryption scheme and Elliptic Curve NRA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ TLS_ECES_ECDSA_WITH_DES_CBC_SHA
Elliptic Curve encryption scheme and Elliptic Curve DSA analog with SHA-1 hashing and 56-bit DES encryption in CBC mode. No export.
- ◆ TLS_DH_anon_WITH_DES_CBC_SHA
Anonymous Diffie-Hellman key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.
- ◆ TLS_DHE_RSA_WITH_DES_CBC_SHA
Ephemeral Diffie-Hellman with RSA key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.
- ◆ TLS_DHE_DSS_WITH_DES_CBC_SHA
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.
- ◆ TLS_DH_RSA_WITH_DES_CBC_SHA
Diffie-Hellman with RSA key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.
- ◆ TLS_DH_DSS_WITH_DES_CBC_SHA
Diffie-Hellman key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.
- ◆ TLS_RSA_WITH_DES_CBC_SHA
RSA key exchange with SHA-1 hashing and DES 56-bit encryption in CBC mode. No export.

40-bit encryption

The following SSL protocols work with 40-bit encryption.

- ◆ TLS_ECDH_ECDSA_EXPORT_WITH_RC4_40_SHA

- Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 40-bit ARC4/RC4 encryption. For export.
- ◆ **SSL_RSA_WITH_RC2_CBC_MD5**
RSA key exchange with MD5 hashing and RC2 40-bit encryption in CBC mode. No export.
 - ◆ **TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA**
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing and 40-bit ARC4/RC4 encryption. For export.
 - ◆ **TLS_ECDH_anon_EXPORT_WITH_DES40_CBC_SHA**
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing and 40-bit DES encryption in CBC mode. For export.
 - ◆ **TLS_ECDHE_ECNR_EXPORT_WITH_RC4_40_SHA**
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 40-bit ARC4/RC4 encryption. For export.
 - ◆ **TLS_ECDHE_ECNR_EXPORT_WITH_DES40_CBC_SHA**
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing and 40-bit DES encryption in CBC mode. For export.
 - ◆ **TLS_ECDHE_ECDSA_EXPORT_WITH_RC4_40_SHA**
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 40-bit ARC4/RC4 encryption. For export.
 - ◆ **TLS_ECDHE_ECDSA_EXPORT_WITH_DES40_CBC_SHA**
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing and 40-bit DES encryption in CBC mode. For export.
 - ◆ **TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA**
Anonymous Diffie-Hellman key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.
 - ◆ **TLS_DH_anon_EXPORT_WITH_RC4_40_MD5**
Anonymous Diffie-Hellman key exchange with MD5 hashing and 40-bit ARC4/RC4 encryption. For export.
 - ◆ **TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA**
Ephemeral Diffie-Hellman with RSA key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.
 - ◆ **TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA**
Ephemeral Diffie-Hellman key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.

- ◆ TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
Diffie-Hellman with RSA key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.
- ◆ TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
Diffie-Hellman key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.
- ◆ TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
RSA key exchange with SHA-1 hashing and DES 40-bit encryption in CBC mode. For export.
- ◆ TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
RSA key exchange with MD5 hashing and RC2 40-bit encryption in CBC mode. For export.
- ◆ TLS_RSA_EXPORT_WITH_RC4_40_MD5
RSA key exchange with MD5 hashing and ARC4/RC4 40-bit encryption. For export.

0-bit encryption

The following SSL protocols work with 0-bit encryption.

- ◆ TLS_ECDH_anon_NULL_WITH_SHA
Elliptic Curve Anonymous Diffie-Hellman with SHA-1 hashing. For export.
- ◆ TLS_ECMQV_ECNRN_NULL_SHA
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve NRA analog with SHA-1 hashing. For export.
- ◆ TLS_ECMQV_ECDSA_NULL_SHA
Elliptic Curve Menezes-Qu-Vanstone and Elliptic Curve DSA key exchange with SHA-1 hashing. For export.
- ◆ TLS_ECDH_ECNRN_NULL_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing. For export.
- ◆ TLS_ECDH_ECDSA_NULL_SHA
Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing. For export.
- ◆ TLS_ECDHE_ECNRN_NULL_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve NRA analog with SHA-1 hashing. For export.

- ◆ TLS_ECDHE_ECDSA_NULL_SHA
Ephemeral Elliptic Curve Diffie-Hellman analog and Elliptic Curve DSA analog with SHA-1 hashing. For export.
- ◆ TLS_ECES_ECNRA_NULL_SHA
Elliptic Curve encryption scheme and Elliptic Curve NRA analog with SHA-1 hashing. For export.
- ◆ TLS_ECES_ECDSA_NULL_SHA
Elliptic Curve encryption scheme and Elliptic Curve DSA analog with SHA-1 hashing. For export.
- ◆ TLS_RSA_WITH_NULL_SHA
RSA key exchange with SHA-1 hashing. For export.
- ◆ TLS_RSA_WITH_NULL_MD5
RSA key exchange with MD5 hashing. For export.
- ◆ TLS_NULL_WITH_NULL_NULL
No key exchange, hashing or encryption. For export.

WebLOAD-supported XML DOM Interfaces

WebLOAD supports the following XML DOM Document Interfaces:

- ◆ XML Document Interface
- ◆ Node Interface
- ◆ Node List Interface
- ◆ NamedNodeMap Interface
- ◆ ParseError Interface
- ◆ Implementation Interface

The tables in this appendix list the properties and methods of the interfaces supported by WebLOAD.

Table B-1: XML Document Interface Properties

Property	Description
<code>doctype</code>	A read-only property that gets the node for the DTD specified for the document. If no DTD was specified, null is returned.
<code>documentElement</code>	A read/write property that gets/sets the root node of the document.
<code>implementation</code>	A read-only property that returns the implementation interface for this document.
<code>parseError</code>	A read-only property that provides an object that summarizes the last parsing error encountered.
<code>preserveWhitespace</code>	A read-write property that informs the parser whether the default mode of processing is to preserve whitespace or not. The default value of this property is <code>false</code> .
<code>readyState</code>	A read-only property indicating the status of instantiating the XML processor and document download.
<code>resolveExternals</code>	A read-write property that informs the parser that resolvable namespaces (a namespaces URI that begin with an “ <code>x-schema:</code> ” prefix), DTD external subsets, and external entity references should be resolved at parse time.
<code>url</code>	A read-only property that returns the canonicalized URL for the XML document specified in the last call to <code>load()</code> .
<code>validateOnParse</code>	A read/write property that turns validation on at parse time if the value of <code>bool</code> is true, off if <code>validate</code> is false.

Table B-2: XML Document Interface Methods

Method	Description
<code>abort()</code>	Aborts an asynchronous download in progress.
<code>createAttribute(name)</code>	Creates a node of type <code>ATTRIBUTE</code> with the name supplied.
<code>CreateCDATASection(data)</code>	Creates a node of type <code>CDATA_SECTION</code> with <code>nodeValue</code> set to data.
<code>createComment(data)</code>	Creates a node of type <code>COMMENT</code> with <code>nodeValue</code> set to data.
<code>createDocumentFragment</code>	Creates a node of type <code>DOCUMENT_FRAGMENT</code> in the context of the current document.
<code>createElement(tagName)</code>	Creates a node of type <code>ELEMENT</code> with the <code>nodeName</code> of <code>tagName</code> .
<code>createEntityReference(name)</code>	Creates a node of type <code>ENTITY_REFERENCE</code> where <code>name</code> is the name of the entity referenced.
<code>CreateNode(type, name, namespaceURI)</code>	Creates a node of the type specified in the context of the current document. Allows nodes to be created as a specified namespace.
<code>CreateProcessingInstruction(target, data)</code>	Creates a node of type <code>PROCESSING_INSTRUCTION</code> with the target specified and <code>nodeValue</code> set to data.
<code>createTextNode(data)</code>	Creates a node of type <code>TEXT</code> with <code>nodeValue</code> set to data.
<code>GetElementsByTagName(tagname)</code>	Returns a collection of all descendent <code>Element</code> nodes with a given <code>tagName</code> .
<code>load(url)</code>	Loads an XML document from the location specified by the <code>url</code> . If the <code>url</code> cannot be resolved or accessed or does not reference an XML document, the <code>documentElement</code> is set to null and an error is returned. Returns a Boolean.
<code>loadXML(xmlstring)</code>	Loads an XML document using the supplied string. <code>xmlstring</code> can be an entire XML document or a well-formed fragment. If the XML within <code>xmlstring</code> cannot be loaded, the <code>documentElement</code> is set to null and an error is returned.
<code>NodeFromID(idstring)</code>	Returns the node that has an <code>ID</code> attribute with the value corresponding to <code>idString</code> .
<code>Save()</code>	Serialize the XML. The parameter can be a filename, an ASP response, an XML Document, or any other COM object that supports <code>IStream</code> , <code>IPersistStream</code> , or <code>IPersistStreamInit</code> .

Table B-3: Node Interface Properties

Property	Description
<code>attributes</code>	A read-only property that returns a <code>NamedNodeMap</code> containing attributes for this node.
<code>BaseName</code>	A read-only property that returns the right-hand side of a namespace qualified name. For example, <code>yyy</code> for the element <code><xxx:yyy></code> . <code>BaseName</code> must always return a non-empty string.
<code>childNodes</code>	A read-only property that returns a <code>NodeList</code> containing all children of the node.
<code>DataType</code>	A read-write property that indicates the node type.
<code>Definition</code>	A read-only property whose value is the node that contains the definition for this node.
<code>FirstChild</code>	A read-only property that returns the first child node. If the node has no children, <code>FirstChild</code> returns null.
<code>LastChild</code>	A read-only property that returns the last child node. If the node has no children, <code>LastChild</code> returns null.
<code>NextSibling</code>	A read-only property that returns the node immediately following this node in the children of this node's parent. Returns null if no such node exists.
<code>NamespaceURI</code>	A read-only property that returns the URI for the namespace (the <code>uuu</code> portion of the namespace declaration <code>xmlns:nnn="uuu"</code>). If there is no namespace on the node that is defined within the context of the document, "" is returned.
<code>nodeName</code>	A read-only property indicating the name of the node.
<code>NodeType</code>	A read-only property indicating the type of node.
<code>NodeTypeString</code>	Returns the node type in string form.
<code>NodeTypedValue</code>	A read/write property for the typed value of the node.
<code>NodeValue</code>	A read/write property for the value of the node.
<code>OwnerDocument</code>	A property that indicates the document to which the node belongs or when the node is removed from a document.
<code>parentNode</code>	A read-only property that provides a pointer to the parent.
<code>parsed</code>	A read-only property that indicates that this node and all of its descendants have been parsed and instantiated. This is used in conjunction with asynchronous access to the document.

Table B-3: Node Interface Properties

Property	Description
<code>prefix</code>	A read-only property that returns the prefix specified on the element, attribute of entity reference. For example, <code>xxx</code> for the element <code><xxx:yyy></code> . If there is no prefix specified, <code>""</code> is returned.
<code>previousSibling</code>	A read-only property that returns the node immediately preceding this node in the children of this node's parent. Returns null if no such node exists.
<code>specified</code>	A read-only property indicating the node was specified directly in the XML source and not implied by the DTD schema.
<code>text</code>	A string representing the content of the element and all descendants. For example "content of tag" in <code><someTag size=34> content of tag </someTag></code> .
<code>xml</code>	A read-only property that returns the XML representation of the node and all its descendants as a string.

Table B-4: Node Interface Methods

Method	Description
<code>appendChild(newChild)</code>	A method to append <code>newChild</code> as the last child of this node.
<code>cloneNode(deep)</code>	A method to create a new node that is an exact clone (same name, same attributes) as this node. When <code>deep</code> is false, only the node and attributes without its children are cloned. When <code>deep</code> is true, the node and all its descendants are cloned.
<code>hasChildNodes()</code>	A method that indicates whether the node has children.
<code>InsertBefore(newChild, oldChild)</code>	A method to insert <code>newChild</code> as a child of this node. <code>oldChild</code> is returned. <code>oldChild</code> must be a child node of the element, otherwise an error is returned. If <code>newChild</code> is null, the <code>oldChild</code> is removed.
<code>removeChild(child)</code>	A method to remove a <code>childNode</code> from a node. If <code>childNode</code> is not a child of the node, an error is returned.
<code>ReplaceChild(newChild, oldChild)</code>	A method to replace <code>oldChild</code> with <code>newChild</code> as a child of this node.

Table B-4: Node Interface Methods

Method	Description
<code>selectNodes(query)</code>	Returns a <code>NodeList</code> containing the results of the query indicated by <code>query</code> , using the current node as the query context. If no nodes match the query, an empty <code>NodeList</code> is returned. If there is an error in the query string, the DOM error reporting is used.
<code>SelectSingleNode(query)</code>	Returns a single node that is the first node in the <code>NodeList</code> returned from the query, using the current node as the query context. If no nodes match the query, null is returned. If there is an error in the query string, an error is returned.
<code>TransformNode(stylesheetsDOMNode)</code>	Returns the results of processing the source <code>DOMNode</code> and its children with the stylesheet indicated by <code>stylesheetDOMNode</code> . The source defines the entire context on which the stylesheet operates, so ancestor or id navigation outside of the scope is not allowed. The stylesheet parameter must be either a DOM Document node, in which case the document is assumed to be an ASL stylesheet, or a DOM Node in the <code>xsl</code> namespace, in which case this node is treated as a standalone.
<code>TransformNodeToObject(stylesheets, Object)</code>	Sends the results of the transform to the requested object, either in <code>IStream</code> or a DOM Document.

Table B-5: Node List Interface

Property	Description
<code>length</code>	The number of nodes in the <code>NodeList</code> . The length of the list will change dynamically as children or attributes are added/deleted from the element.
<code>nextNode</code>	Returns the next node in the <code>NodeList</code> based on the current node.
Method	
<code>item(index)</code>	Returns the node in the <code>NodeList</code> with the specified index.
<code>reset()</code>	Returns the iterator to the uninstantiated state; that is, before the first node in the <code>NodeList</code> .

Table B-6: NamedNodeMap Interface

Property	Description
<code>length</code>	The number of nodes in the <code>NamedNodeMap</code> . The length of the list will change dynamically as children or attributes are added/deleted from the element.
Method	
<code>getNamedItem(name)</code>	Returns the node corresponding to the attribute with name. If name is not an attribute, null is returned.
<code>GetQualifiedItem(baseName, namespaceURI)</code>	Allows the specification of a qualifying namespaceURI to access a namespace qualified attribute. It returns the node corresponding to the attribute with baseName in the namespace specified by namespaceURI. If the qualified name (baseName+namespaceURI) is not an attribute, null is returned.
<code>item(index)</code>	Returns the node in the <code>NamedNodeMap</code> with the specified index. If the index is greater than the total number of nodes, null is returned. If the index is less than zero, null is returned.
<code>nextnode()</code>	Returns the next node in the <code>NodeList</code> based on the current node.
<code>RemoveNamedItem(name)</code>	Removes the attribute node corresponding to name and returns the node. If name is not an attribute, null is returned.
<code>RemoveQualifiedItem(baseName, namespaceURI)</code>	Removes the namespaceURI qualified attribute node corresponding to baseName and returns the node. If the qualified name is not an attribute, null is returned.
<code>reset()</code>	Returns the iterator to the uninstiated state; that is before the first node in the <code>NodeList</code> .
<code>SetNamedItem(namedItem)</code>	Adds the attribute Node to the list. If an attribute already exists with the same name as that specified by nodeName of DOMNode, the attribute is replaced and the node is returned. Otherwise, Node is returned.

Table B-7: ParseError Interface

Item	Description
<code>errorCode</code>	Returns the error code number in decimal.
<code>filepos</code>	Returns the absolute file position where the error occurred.
<code>line</code>	Returns number of the line containing the error.
<code>linepos</code>	Returns the character position where the error occurred.
<code>reason</code>	Returns the reason for the error.
<code>srcText</code>	Returns the full text of the line containing the error.
<code>url</code>	Returns the URL of the XML file containing the error.

Table B-8: Implementation Interface

Item	Description
<code>HasFeature</code> (<code>feature</code> , <code>version</code>)	The method returns true if the specified version of the parser supports the specified feature. In Level 1, “1.0” is the only valid version value.

WebLOAD Internet Protocols Reference

This chapter provides detailed reference information on WebLOAD support for the following Internet protocols:

- ◆ FTP, through the *wFTP Object*, description beginning on page 400
- ◆ IMAP, through the *wIMAP Object*, description beginning on page 413
- ◆ NNTP, through the *wNNTP Object*, description beginning on page 425
- ◆ POP, through the *wPOP Object*, description beginning on page 436
- ◆ SMTP, through the *wSMTP Object*, description beginning on page 444
- ◆ TCP, through the *wTCP Object*, description beginning on page 451
- ◆ Telnet, through the *wTelnet Object*, description beginning on page 457
- ◆ UDP, through the *wUDP Object*, description beginning on page 463

wlFTP Object

The `wlFTP` object provides support for FTP (File Transfer Protocol) load and functional testing within WebLOAD. Support for standard FTP operation is included. FTP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Logon()` method; otherwise an exception is thrown.

To access the `wlFTP` object, you must include the `wlFtp.js` file in your `InitAgenda()` function.

wlFTP Properties

Data

The `Data` property lets you specify the local data stream to upload to the host. You use this property to upload data. For example:

```
ftp.Data = datastream
```

DataFile

The `DataFile` property lets you specify the local file to upload to the host. For example:

```
ftp.DataFile = filename
```

document

The `document` property is an array containing the files downloaded and uploaded in the current FTP session. For example:

```
var recentdownload = ftp.document[1]
```

Outfile

The `Outfile` property lets you specify the name of a downloaded file. You use this property to rename a downloaded file as it is transferred to your computer. This property must be an explicit file name, not a pattern. If you specify the `Outfile` property, the `document`

property remains empty. If you are downloading a series of files, only the last file downloaded is stored in the `Outfile`.

If you want to store all of the files downloaded, either delete the `Outfile` property or specify an empty value. The downloaded files are then stored in the document property. For example:

```
ftp.Outfile = filename
```

PassiveMode

The `PassiveMode` property lets you use FTP through firewalls. Valid values are:

- ◆ True - passive mode is set, and you may FTP through firewalls.
- ◆ False - active mode is set, and you may not FTP through firewalls.

For example:

```
ftp.PassiveMode = modesetting
```

PassWord

The `PassWord` property lets you specify a password when logging on to a host. You use this property to log onto a restricted FTP host. WebLOAD automatically sends the password to the FTP host when a `wFTP` object connects to an FTP host.

```
ftp.PassWord = password
```

Caution: The password appears in plain text in the Agenda. The password is visible to any user who has access to the Agenda.

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption.

```
var filesize = ftp.Size
```

StartByte

The `StartByte` property lets you specify the byte offset to start transferring from. The default value is zero. This property automatically resets to zero after each transfer. You use this property to specify a starting point when resuming interrupted transfers.

```
ftp.StartByte = byteoffset
```

TransferMode

The `TransferMode` property lets you specify the transfer mode for uploaded and downloaded files. You must specify the transfer mode before each transfer. If you do not specify a transfer mode, auto, the default mode, is used. Valid values are:

- ◆ auto - 0
- ◆ text - 1
- ◆ binary - 2

You may also specify the transfer mode using the following constants:

- ◆ `WLFtp.TMODE_ASCII` - text
- ◆ `WLFtp.TMODE_BINARY` - binary
- ◆ `WLFtp.TMODE_DEFAULT` - auto

For example:

```
ftp.TransferMode = transfermode
```

UserName

The `UserName` property lets you specify a User ID when logging on to a host. You use this property to log onto a restricted FTP host. WebLOAD automatically sends the user name to the FTP host when a `wlFTP` object connects to an FTP host.

```
ftp.UserName = username
```

wlFTP Methods

Append()

Syntax	<code>Append(<i>pattern</i>)</code>
Parameters	
<code>pattern</code>	The file to which you are appending. This may be a specific file name, or it may contain wildcards.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Similar to the <code>Upload()</code> method, <code>Append()</code> adds the data to the target file instead of overwriting it. If the target file does not exist, <code>Append()</code> creates it.

AppendFile()

Syntax	<code>AppendFile (filename)</code>
Parameters	
<code>filename</code>	The remote file to which you want to append data.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Uploads a local file and appends it to the specified file on the host. The local file is specified by the DataFile property. The destination file is specified by the <code>filename</code> parameter. If the DataFile property is not specified, then the contents of the Data property are sent as a datastream to be appended to the file specified by the <code>filename</code> parameter. If the target file does not exist, AppendFile() creates it.

ChangeDir()

Syntax	<code>ChangeDir (name)</code>
Parameters	
<code>name</code>	The name of the directory to which you want to move.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Changes the current working directory on the host to the one specified by the <code>name</code> parameter.

ChFileMod()

Syntax	<code>ChFileMod (filename , attributes)</code>
Parameters	
<code>filename</code>	The name of the file you want to alter. This parameter may be a specific file name, or it may contain wildcards.
<code>attributes</code>	The new attributes assigned to the file. Values are specified in the three digit 0-7 format.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Changes attributes of the specified file according to the values specified in the <code>attribute</code> parameter.

ChMod()

Syntax	<code>ChMod(<i>pattern</i>, <i>attributes</i>)</code>
Parameters	
<i>pattern</i>	The name of the files and directories you want to alter. This parameter may be a specific file name, or it may contain wildcards.
<i>attributes</i>	The new attributes assigned to the file. Values are specified in the three digit 0-7 format.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Uses a loop to changes attributes of the specified files and directories according to the values specified in the <i>attribute</i> parameter. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files unchanged. To avoid this risk you must write your own loop with error handling capability.

Delete()

Syntax	<code>Delete(<i>pattern</i>)</code>
Parameters	
<i>pattern</i>	The file you are deleting. This may be a specific file name, or it may contain wildcards.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the specified files from the FTP host. This function calls the <code>DeleteFile()</code> method in a loop to delete all the specified files. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files undeleted.

DeleteFile()

Syntax	<code>Delete(<i>filename</i>)</code>
Parameters	
<i>filename</i>	The file you are deleting. This must be a specific file name.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the specified file from the FTP host.

Dir()

Syntax	<code>Dir(<i>pattern</i>)</code>
Parameters	
<i>pattern</i>	The name of the file or directory for which you are searching. This may be a specific file name, or it may contain wildcards.
Return Value	Returns a JavaScript array with the following members if successful, an exception if unsuccessful. <pre> a[].fileName // name of file a[].fileAttributes // attribute string a[].fileTime // date and time of last modification a[].fileSize // size of file in bytes a[].isDir // 1 if the entry represents a // directory, 0 for a file </pre> <p>Note: If the host supports only basic information, only the <code>fileName</code> property of the array is defined.</p>
Comments	Lists files and directories that match the <i>pattern</i> parameter in the current directory of the host. This method returns detailed information if the server supports it.

Download()

Syntax	<code>Download(<i>pattern</i>)</code>
Parameters	
<i>pattern</i>	The file you are downloading. This may be a specific file name, or it may contain wildcards.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Uses a loop to download the specified files to the local computer. If the <code>property</code> has been set, the data is saved to the specified file. If the <code>Outfile</code> property has not been set, the file is saved with its current name. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files not downloaded.

DownloadFile()

Syntax	<code>Download(<i>filename</i>)</code>
Parameters	
<i>filename</i>	The file you are downloading. This must be a specific file name.
Return Value	Null if successful, an exception if unsuccessful.

Comments	Downloads a file to the local computer. If the <code>Outfile</code> property has been set, the data is saved to the specified file. If the <code>Outfile</code> property has not been set, the file is saved with its current name.
-----------------	---

GetCurrentPath()

Syntax	<code>GetCurrentPath()</code>
Parameters	None
Return Value	A string containing the current path if successful, an exception if unsuccessful.
Comments	Returns the current path on the host.

GetStatusLine()

Syntax	<code>GetStatusLine()</code>
Parameters	None
Return Value	A string containing the current path if successful, an exception if unsuccessful.
Comments	A string containing the latest response string if successful, an exception if unsuccessful.

ListLocalFiles()

Syntax	<code>ListLocalFiles(<i>filter</i>)</code>
Parameters	
<i>filter</i>	The files you want to list. The filter may be a pattern or a specific file name.
Return Value	An array of matching objects with following properties if successful, an exception if unsuccessful. <pre> a[].fileName // A string containing name of the file a[].isDir // A Boolean, true if the entry represents a directory </pre>
Comments	Lists files matching the <code>filter</code> parameter in the current directory of the local computer.

Logoff()

Syntax	<code>Logoff ()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the FTP host.

Logon()

Syntax	<code>Logon (<i>host</i> , [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
<i>port</i>	The port to which you are connecting. If you do not specify a port, the default FTP port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts a conversation with the FTP host. If you are logging on to a restricted site, you must have specified the <code>UserName</code> and <code>PassWord</code> properties before using this method.

MakeDir()

Syntax	<code>MakeDir (<i>name</i>)</code>
Parameters	
<i>name</i>	The name of the new directory that you are creating.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Creates a new directory beneath the current directory on the host.

RemoveDir()

Syntax	<code>RemoveDir (<i>name</i>)</code>
Parameters	
<i>name</i>	The name of the directory that you are deleting.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the named directory from the host.

Note: You may not delete a directory until that directory is empty. Remove all files from the directory before using the `RemoveDir()` method. You may use the `Delete()` method to delete files on the host.

Rename()

Syntax	<code>Rename(<i>from</i>, <i>to</i>)</code>
Parameters	
<i>from</i>	The file that you want to rename.
<i>to</i>	The new file name for the file. If this file already exists, it is overwritten.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Renames the files in the current directory described by the <i>from</i> parameter to the name described in the <i>to</i> parameter.

SendCommand()

Syntax	<code>SendCommand(<i>string</i>)</code>
Parameters	
<i>string</i>	The string you are sending to the host.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Sends a string to the host without modification. This method is useful for interacting directly with the host using non-standard or unsupported extensions.

Upload()

Syntax	<code>Upload(<i>pattern</i>)</code>
Parameters	
<i>pattern</i>	The file you are uploading. This may be a specific file name, or it may contain wildcards.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Uses a loop to upload the local files specified by the <i>pattern</i> parameter to the host. The file is not renamed, and values specified by the <code>DataFile</code> and <code>Data</code> property are ignored. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files not transported.

UploadFile()

Syntax	<code>UploadFile(<i>filename</i>)</code>
Parameters	
<i>filename</i>	The destination name of the local file. This parameter may be the same name as the local file name, or it may be used to rename the file once it arrives at the host.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Uploads a local file to the host. The local file is specified by the <code>DataFile</code> property. The destination file name is specified by the <i>filename</i> parameter. If the <code>DataFile</code> property is not specified, then the contents of the <code>Data</code> property are sent as a datastream to be saved under the name specified by the <i>filename</i> parameter.

UploadUnique()

Syntax	<code>UploadUnique()</code>
Parameters	None
Return Value	A string containing the name of the newly created file if successful, an exception if unsuccessful.
Comments	Uploads data or a file to a newly created, unique file on the host. The file name is created by the host, and returned as a string value. The local file is specified by the <code>DataFile</code> property. If the <code>DataFile</code> property is not specified, then the contents of the <code>Data</code> property are sent as a datastream.

WLFTP()

Syntax	<code>new WLFTP()</code>
Parameters	None
Return Value	A new wFTP object.
Comments	Creates a new wFTP object, used to interact with the server.
Example	<pre>function InitClient() { myNewFtpObject = new WLFTP() }</pre>

FTP Sample Code

```
// Agenda Initialization
function InitAgenda() {
    // include the file that enables FTP
    IncludeFile("wlFtp.js",WLExecuteScript)
}

function InitClient(){
    // Create the FTP object we need to interact with the server
    ftp=new WLFtp()
}

function TerminateClient(){
    // Delete the FTP object we used
    delete ftp
}
//=====

//Body Of Agenda. Give user name and password and login
ftp.UserName="UserID" // Set the user name
ftp.PassWord="TopSecret" // Set the password
//this.PassiveMode=true; // Enable this if firewall is in the way
ftp.Logon("localhost") // Login to the server
//=====

//Test Download
ftp.TransferMode = ftp.TMODE_ASCII; // Force all downloads ASCII
ftp.Outfile="c:\\downloaded.txt";
    // Define a local file to save the downloaded file
ftp.Download("file.txt"); // Grab the remote file
// The remote file may be a wildcard, so for each file
// downloaded an entry is made in the document array.
// With this approach an Outfile is not required. Instead the
// document object holds the downloaded files for this client.
// The loop below loops through each entry in the document
// array and writes the file contents out to the log
for (var i = 0; i < ftp.document.length; i++)
{
    InfoMessage(ftp.document[i]);
}
//=====

//Test Upload
ftp.TransferMode = ftp.TMODE_ASCII;
ftp.DataFile="c:\\upload.txt";
    // define local file to upload
ftp.UploadFile("hello.txt");
    // upload it to the remote host as "hello.txt"
```

```

ftp.Data="hello world";
           // define a string to send to the remote host
ftp.UploadFile("hello.txt");
           // upload the string and save it as "hello.txt"
//=====

//Test Append
ftp.TransferMode = ftp.TMODE_ASCII;
ftp.DataFile="c:\\append.txt";
           // identify a local file to upload
ftp.AppendFile("hello.txt");
           // add it to the existing contents of "hello.txt"
//=====

//Test Delete
ftp.Delete("hello.txt");
           // delete "hello.txt" from the remote server
//=====

//Test Directory Functions
ftp.MakeDir("DirectoryName");           // make a new directory
ftp.ChangeDir("DirectoryName");        // change to that directory
ftp.DataFile="c:\\file1.txt";          // select a local file
ftp.Upload("file1.txt");               // upload it to the new directory
var files = ftp.Dir("*.");
           // Generate a list of the files in that directory
for (var i = 0 ; i < files.length; i++)
{
    InfoMessage("the file name is:" + files[i].fileName);
           // Print each file's name to the log
}
ftp.Delete("*.");                       // delete the files on the directory
ftp.ChangeDir("..");                     // go up a level in the tree
ftp.RemoveDir("DirectoryName");        // delete the directory itself
//=====

//Test Advanced Directory Handling
var files = ftp.Dir("*.txt");           // show all the text files
if (files.length > 0)
           // IF there are any entries to go through
{
           // THEN print their detailed attributes to the log
    for (var i = 0 ; i < files.length; i++)
    {
           // Print each file's details to the log
        InfoMessage(files[0].fileName);           // name
        InfoMessage(files[0].fileAttributes);    // attributes
        InfoMessage(files[0].fileTime);          // timestamp
        InfoMessage(files[0].fileSize);          // size in bytes
    }
}

```

```
        InfoMessage(files[0].dirFlag);
                        // set when the object is a directory
    }
}
//=====

//Test General Functions
status = ftp.GetStatusLine();
                        // what was the last response from the server?
InfoMessage("status= "+ status);           // print it
path = ftp.GetCurrentPath();               // where am I?
InfoMessage("path="+ path);              // right here
//=====

catch (e)
{
    InfoMessage ("Error" + e)
}

ftp.Logoff()           // do not forget to log out of the session
InfoMessage("done")   // this is the end of the FTP sample script
```


wIIMAP Object

The `wIIMAP` object provides support for IMAP4 (Internet Message Access Protocol) load and functional testing within WebLOAD. Support for standard IMAP operation is included. IMAP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect()` method; otherwise an exception is thrown.

To access the `wIIMAP` object, you must include the `wIImap.js` file in your `InitAgenda()` function.

wIIMAP Properties

CurrentMessage

The `CurrentMessage` property returns the number of the current message. You use this property to track the current message in relation to other messages on the host. For example:

```
var currentmessagenumber = imap.CurrentMessage
```

CurrentMessageID

The `CurrentMessageID` property returns the ID of the current message. You use this property to track the current message in relation to other messages on the host. For example:

```
var messagenumber = imap.CurrentMessageID
```

document

The `document` property is an object with four properties:

- ◆ `Headers` - A string containing the header of the message
- ◆ `MessageText` - A string containing the text of the message
- ◆ `Size` - An integer describing the size of the message in bytes
- ◆ `Attachments` - An array of objects, with each attachment existing as an object with the following properties:
 - `contentencoding` - The encoding of the attachment

- `contenttype` - The content type of the attachment
- `filename` - The file name of the attachment
- `messagetext` - The text of the attachment
- `partname` - The part name of the message
- `size` - The size of the attachment in bytes

For example:

```
var recentdocument = imap.document
var messageheaders = recentdocument.MessageHeaders
var messagetext = recentdocument.MessageText
var messagesize = recentdocument.MessageSize
var messageattachments = recentdocument.attachments
```

Mailbox

The `Mailbox` property specifies the name of the mailbox with which you want to interact. You use this property to create, edit, and delete mailboxes. For example:

```
imap.Mailbox = mailboxname
```

MaxLines

The `MaxLines` property lets you specify the maximum number of lines per email to retrieve from an IMAP host. You use this property to specify the number of lines to retrieve from each email. For example:

```
imap.Maxlines = numberoflines
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save a file or message locally on your computer. When you write to the `Outfile`, you overwrite the existing content. To avoid overwriting the existing content, you must specify a new `Outfile` each time you write. For example:

```
imap.Outfile = filename
```

PassWord

The `PassWord` property lets you specify a password when logging on to a host. You use this property to log onto a restricted IMAP host. WebLOAD automatically sends the password to the IMAP host when a wIIMAP object connects to an IMAP host. For example:

```
imap.PassWord = password
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = imap.Size
```

UserName

The `UserName` property lets you specify a User ID when logging on to a host. You use this property to log onto a restricted IMAP host. WebLOAD automatically sends the user name to the IMAP host when a wIIMAP object connects to an IMAP host. For example:

```
imap.UserName = username
```

wIIMAP Methods

Connect()

Syntax	<code>Connect(<i>host</i>, [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
<i>port</i>	The port to which you are connecting. If you do not specify a port, the default IMAP port (port 143) is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts an IMAP session with the host. When you connect, you are connecting to a specific mailbox within the host, as specified by your User ID.

CreateMailbox()

Syntax	CreateMailbox()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Creates the mailbox specified in the Mailbox property. The created mailboxes continue to exist after the end of the Agenda. To remove a mailbox, use the DeleteMailbox() method.

Delete()

Syntax	Delete([MessageSet])
Parameters	
<i>MessageSet</i>	The identifier of the message you want to delete. You may specify a single message number, or you may specify a range, separated by a colon. For example, 1:10 deletes messages one through ten. If you do not specify a message ID, the current message is deleted.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the message with the corresponding ID. If no ID is specified, then the current message is deleted.

DeleteMailbox()

Syntax	DeleteMailbox()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the mailbox specified in the Mailbox property.

Disconnect()

Syntax	Disconnect()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the IMAP host.

GetMessageCount()

Syntax	<code>GetMessageCount ()</code>
Parameters	None
Return Value	A string containing the number of messages on the host if successful, an exception if unsuccessful.
Comments	Returns the number of messages waiting on the host.

GetStatusLine()

Syntax	<code>GetStatusLine ()</code>
Parameters	None
Return Value	A string containing the latest response string if successful, an exception if unsuccessful.
Comments	Returns the latest response string from the host.

ListMailboxes()

Syntax	<code>ListMailboxes (pattern)</code>
Parameters	<i>pattern</i>
	The mailbox that you want to appear in the list. This may be a specific name, or it may contain wildcards.
Return Value	A string listing the matching mailboxes if successful, an exception if unsuccessful.
Comments	Lists mailboxes matching the <i>pattern</i> parameter.

RecentMessageCount()

Syntax	<code>RecentMessageCount ()</code>
Parameters	None
Return Value	A string containing the number of new messages on the host if successful, an exception if unsuccessful.
Comments	Returns the number of new messages waiting on the host.

RenameMailbox()

Syntax	<code>RenameMailbox(<i>string</i>)</code>
Parameters	
<i>string</i>	The new name for the mailbox.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Renames the mailbox specified in the Mailbox property.

Retrieve()

Syntax	<code>Retrieve([<i>MessageSet</i>])</code>
Parameters	
<i>MessageSet</i>	The identifier of the message you want to retrieve. You may specify a single message number, or you may specify a range, separated by a colon. For example, 1:10 returns messages one through ten. If you do not specify a message ID, the next message is returned.
Return Value	A document for each message specified if successful, an exception if unsuccessful.
Comments	Returns the message with the corresponding ID. If no ID is specified, then the next message is returned.

Search()

Syntax	<code>Search(<i>string</i>)</code>
Parameters	
<i>string</i>	

The criteria for your search. Valid values are:

- ALL - All messages in the mailbox - this is the default initial key for AND-ing.
- ANSWERED - Messages with the \Answered flag set.
- BCC - Messages that contain the specified string in the envelope structure's BCC field.
- BEFORE - Messages whose internal date is earlier than the specified date.
- BODY - Messages that contain the specified string in the body of the message.
- CC - Messages that contain the specified string in the envelope structure's CC field.
- DELETED - Messages with the \Deleted flag set.
- DRAFT - Messages with the \Draft flag set.
- FLAGGED - Messages with the \Flagged flag set.
- FROM - Messages that contain the specified string in the envelope structure's FROM field.
- HEADER - Messages that have a header with the specified field-name (as defined in) and that contains the specified string in the field-body.
- KEYWORD - Messages with the specified keyword set.
- LARGER - Messages with an size larger than the specified number of octets.
- NEW Messages that have the \Recent flag set but not the \Seen flag. This is functionally equivalent to "(RECENT UNSEEN)".
- NOT - Messages that do not match the specified search key.
- OLD - Messages that do not have the \Recent flag set. This is functionally equivalent to "NOT RECENT" (as opposed to "NOT NEW").
- ON - Messages whose internal date is within the specified date.
- OR - Messages that match either search key.
- RECENT - Messages that have the \Recent flag set.
- SEEN - Messages that have the \Seen flag set.
- SENTBEFORE - Messages whose Date: header is earlier than the specified date.
- SENTON - Messages whose Date: header is within the specified date.
- SENTSINCE - Messages whose Date: header is within or later than the specified date.
- SINCE - Messages whose internal date is within or later than the specified date.
- SMALLER - Messages with an RFC822.SIZE smaller than the specified number of octets.
- SUBJECT - Messages that contain the specified string in the envelope structure's SUBJECT field.
- TEXT - Messages that contain the specified string in the header or body of the message.

- TO - Messages that contain the specified string in the envelope structure's TO field.
- UID - Messages with unique identifiers corresponding to the specified unique identifier set.
- UNANSWERED - Messages that do not have the \Answered flag set.
- UNDELETED - Messages that do not have the \Deleted flag set.
- UNDRAFT - Messages that do not have the \Draft flag set.

Return Value	A string containing the IDs of messages that meet the search criteria if successful, an exception if unsuccessful.
Comments	Searches the current mailbox for messages meeting the specified search criteria.

SendCommand()

Syntax	<code>SendCommand(<i>string</i>)</code>
Parameters	
<i>string</i>	The string you are sending to the host.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Sends a string to the host without modification. This method is useful for interacting directly with the host using non-standard or unsupported extensions.

SubscribeMailbox()

Syntax	<code>SubscribeMailbox()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Subscribes to the mailbox specified in the <code>Mailbox</code> property.

UnsubscribeMailbox()

Syntax	<code>UnsubscribeMailbox()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Unsubscribes from the mailbox specified in the <code>Mailbox</code> property.

WLImap()

Syntax	<code>new WLImap()</code>
Parameters	None
Return Value	A new wIIMAP object.
Comments	Creates a new wIIMAP object, used to interact with the server.
Example	<pre>function InitClient() { myNewImapObject = new WLImap() myNewImapObject.Connect("HostName") }</pre>

IMAP Sample Code

```

// Agenda Initialization
function InitAgenda() {
    IncludeFile("wlImap.js",WLExecuteScript)
}

function InitClient() {
    imap=new WLImap() // create the new IMAP object
// imap.Connect("HostName"); // connect to the server
}

function TerminateClient() {
    imap.Disconnect(); // logout from the server
    delete imap // delete the IMAP object
}
//=====

// Body Of Agenda
InfoMessage("Speed: "+wlGlobals.ConnectionSpeed)
wlGlobals.Debug=1;
imap.UserName="UserID";
imap.Password="TopSecret";
imap.Mailbox="Inbox";
imap.Connect("00.0.0.00");
//=====

//Test Retrieve
/*imap.Retrieve("100");
for (var i = 0; i < imap.wlSource.length; i++)
{
    InfoMessage(imap.wlSource[i]);
    InfoMessage(imap.document.length);
    InfoMessage(imap.document[i].headers);
    InfoMessage(imap.document[i].messageText);
    InfoMessage(imap.document[i].size);
    InfoMessage(imap.document[i].attachments.length);
    for (var j = 0; j < imap.document[i].attachments.length; j++)
    {
        InfoMessage(imap.document[i].attachments[j].contentEncoding);
        InfoMessage(imap.document[i].attachments[j].contentType);
        InfoMessage(imap.document[i].attachments[j].filename);
        InfoMessage(imap.document[i].attachments[j].messageText);
        InfoMessage(imap.document[i].attachments[j].partName);
        InfoMessage(imap.document[i].attachments[j].size);
    }
}*/
//=====

```

```

//Test Delete
imap.Mailbox="Inbox";
InfoMessage(imap.GetMessageCount());
imap.Mailbox="Inbox";
imap.Delete("2");
imap.Mailbox="Inbox";
InfoMessage(imap.GetMessageCount());
//=====

//Test Mailbox Functions:
//      list mailboxes, create mailbox, and then list again
/*InfoMessage("mailboxes are:")
var v1 = imap.ListMailboxes();
for(var i=0; i < v1.length; i++)
    InfoMessage(v1[i]);
imap.Mailbox="mailboxname";
imap.CreateMailbox();
InfoMessage("mailboxes are:")
var v1 = imap.ListMailboxes();
for(var i=0; i < v1.length; i++)
    InfoMessage(v1[i]);
*/
//=====
//      subscribe mailbox, list all subscribed mailboxes
//imap.Mailbox="mailboxname";
//imap.SubscribeMailbox();
/*InfoMessage("subscribed mailboxes are:")
var v2 = imap.ListSubscribedMailboxes();
for(var j=0; j < v2.length; j++)
{
    InfoMessage(v2[j]);
    imap.Mailbox=v2[j];
}*/
//=====
//      list subscribed mailboxes,unsubscribe mailbox,
//      and then list all subscribed mailboxes again
/*InfoMessage("subscribed mailboxes are:")
var v2 = imap.ListSubscribedMailboxes();
for(var j=0; j < v2.length; j++)
{
    InfoMessage(v2[j]);
    imap.Mailbox=v2[j];
}
imap.Mailbox="mailboxname";
imap.UnsubscribeMailbox();
InfoMessage("subscribed mailboxes are:")
var v2 = imap.ListSubscribedMailboxes();
for(var j=0; j < v2.length; j++)
{

```

```

        InfoMessage(v2[j]);
        imap.Mailbox=v2[j];
    }
    */
    //=====
    //
    //                                list mailboxes, rename mailbox,
    //                                and then list mailboxes again
    /*InfoMessage("mailboxes are:")
    var v1 = imap.ListMailboxes();
    for(var i=0; i < v1.length; i++)
        InfoMessage(v1[i]);
    imap.Mailbox="boxname";
    imap.RenameMailbox("newName");
    InfoMessage("mailboxes are:")
    var v1 = imap.ListMailboxes();
    for(var i=0; i < v1.length; i++)
        InfoMessage(v1[i]);
    */
    //=====
    //
    //                                get number of messages from a mailbox
    /*imap.Mailbox="main";
    InfoMessage(imap.GetMessageCount());
    imap.Mailbox="Inbox";
    InfoMessage(imap.GetRecentMessageCount());
    */
    //=====
    //
    //                                delete mailbox and list all the mailboxes
    /*imap.Mailbox="mailboxname";
    imap.DeleteMailbox();
    InfoMessage("subscribed mailboxes are:")
    var v2 = imap.ListSubscribedMailboxes();
    for(var j=0; j < v2.length; j++)
    {
        InfoMessage(v2[j]);
        imap.Mailbox=v2[j];
    }
    */
    //=====
    //
    //                                search
    /*imap.Mailbox="Inbox";
    var found = imap.Search("CC user@address.com");
    InfoMessage("found:")
    for(var j=0; j < found.length; j++)
    {
        InfoMessage(found[j]);
    }
    catch (e)
    {
        InfoMessage ("Error" + e)
    }
    */

```

```
*/  
//=====
```

imap.Disconnect();
delete imap
InfoMessage("done")

wINNTP Object

The `wINNTP` (Network News Transfer Protocol) object provides support for NNTP load and functional testing within WebLOAD. Support for standard NNTP operation is included. NNTP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect()` method; otherwise an exception is thrown.

You must include `catch` and `try` functions in your script to handle exceptions when using the `wINNTP` object. If you do not, the object may cause your Agenda to freeze. A sample `catch` appears in the NNTP code sample at the end of this section.

To access the `wINNTP` object, you must include the `wINntp.js` file in your `InitAgenda()` function.

wINNTP Properties

ArticleText

The `ArticleText` property lets you specify the text appearing in the body of your article. You use this property to write the text of the article itself. For example:

```
nntp.ArticleText = articlecontent
```

Attachments

The `Attachments` property lets you specify an attachment to a posting. The `filename` variable should contain the name of the local file or datastream that you want to attach to the posting. For example:

```
nntp.Attachments = filename
```

AttachmentsEncoding

The `AttachmentsEncoding` property lets you specify the type of encoding you are applying to a message attachment. This property must be specified for each attachment. Valid values are:

- ◆ `7Bit`

- ◆ Quoted
- ◆ Base64
- ◆ 8Bit
- ◆ 8BitBinary

You may also specify the encoding using the following constants:

- ◆ `WLNntp.ENC_7BIT` - 7bit encoding
- ◆ `WLNntp.ENC_QUOTED` - Quoted Printable encoding
- ◆ `WLNntp.ENC_BASE64` - Base64 encoding
- ◆ `WLNntp.ENC_8BIT` - 8Bit encoding
- ◆ `WLNntp.ENC_8BITBINARY` - Binary encoding

For example:

```
nntp.AttachmentsEncoding = encodingtype
```

AttachmentsTypes

The `AttachmentsTypes` property lets you specify the type of attachment you are including in a posting. This property must be specified for each attachment. Valid values are:

- ◆ `True` - specifies a type of file
- ◆ `False` - specifies a type of data

For example:

```
nntp.AttachmentsTypes = typeofattachment
```

document

The `Document` property is an object with two properties. One is a string, `MessageText` containing the text of the article, and the other is an array containing the article attachments and headers. For example:

```
var recentdocument = nntp.document  
var messagetext = recentdocument.MessageText  
var messageattachments = recentdocument.attachments  
var firstattachment = messageattachments[0]  
var secondattachment = messageattachments[1]
```

From

The `From` property lets you describe the Reply To in plain language. You may use this property to identify your Reply To email address in a plain language format. For example:

```
nntp.From = replyname
```

Group

The `Group` property specifies the article group with which you are interacting. You use this to limit searches, posts, and other activities to a specific group. For example:

```
nntp.Group = groupname
```

MaxHeadersLength

The `MaxHeadersLength` property lets you specify the maximum length for headers in a article. You use this property to prevent line folding. For example:

```
nntp.MaxHeadersLength = headersize
```

Organization

The `Organization` property identifies the affiliation of the author. You use this property to identify your professional or personal affiliation. For example:

```
nntp.Organization = organizationname
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save a file or article locally on your computer. For example:

```
nntp.Outfile = filename
```

PassWord

The `PassWord` property lets you specify a password when logging on to a host. You use this property to log onto a restricted NNTP host. WebLOAD automatically sends the password to the NNTP host when a wINNTP object connects to an NNTP host. For example:

```
nntp.PassWord = password
```

Caution: The password appears in plain text in the Agenda. The password is visible to any user who has access to the Agenda

References

The `References` property lets you specify articles that the posted article follows. You use this property to create a thread of related articles. If the resulting reference header is longer than the limit specified in the `MaxHeadersLength` property, it is folded. References must be separated by commas with no spaces in between. For example:

```
nntp.References = article1,article2
```

ReplyTo

The `ReplyTo` property lets you specify the reply address for additional postings. For example:

```
nntp.ReplyTo = replyaddress
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = nntp.Size
```

Subject

The `Subject` property lets you specify the text appearing the subject field of your email. You use this property to provide a brief description of the contents of your article. For example:

```
nntp.Subject = subjectheader
```

To

The `To` property lets you specify the newsgroup to receive your posting. You may specify multiple addresses in a semicolon-separated list. You must specify this property with every article. For example:

```
nntp.To = alt.newsgroup.name; rec.newsgroup.name
```

UserName

The `UserName` property lets you specify a User ID when logging on to a host. You use this property to log onto a restricted NNTP host. WebLOAD automatically sends the user name to the NNTP host when a `wNNTP` object connects to an NNTP host. For example:

```
nntp.UserName = username
```


wINNTP Methods

AddAttachment()

Syntax	<code>AddAttachment(<i>string</i>, <i>type</i>, [<i>encoding</i>])</code>
Parameters	
<i>string</i>	The string you are sending to the host. If you are sending a file, the string is the location and name of the file. If you are sending a data attachment, the string is the data to be attached.
<i>type</i>	The type of attachment you are sending. Valid values are: <ul style="list-style-type: none">• File (default)• Data
[<i>encoding</i>]	The type of encoding to apply to the file. Valid values are: <ul style="list-style-type: none">• 7Bit (default)• Quoted• Base64• 8Bit• 8BitBinary
Return Value	Returns an integer value Attachment ID if successful, an exception if unsuccessful.
Comments	Adds an attachment to the message.

Connect()

Syntax	<code>Connect(<i>host</i>, [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
[<i>port</i>]	The port to which you are connecting. If you do not specify a port, the default NNTP port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts an NNTP session with the host.

DeleteAttachment()

Syntax	DeleteAttachment(<i>string</i>)
Parameters	
<i>string</i>	The ID of the attachment you are deleting.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Removes an attachment from the article.

Disconnect()

Syntax	Disconnect()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the NNTP host.

GetArticle()

Syntax	GetArticle(<i>messageNumber</i>)
Parameters	
<i>messageNumber</i>	The number of the message that you want to retrieve.
Return Value	Null if successful. The article is stored in the document property. An exception if unsuccessful.
Comments	Gets the headers and body of the article specified in the <i>messageNumber</i> parameter for the group specified in the <i>Group</i> property. If the <i>Outfile</i> property is specified, the returned article is stored in the output file as well as in the document property.

GetArticleCount()

Syntax	GetArticleCount()
Parameters	None
Return Value	An integer count of the number of articles in the group if successful, an exception if unsuccessful.
Comments	Returns the number of articles in the group specified by the <i>Group</i> property.

GetStatusLine()

Syntax	<code>GetStatusLine()</code>
Parameters	None
Return Value	A string containing the latest response string if successful, an exception if unsuccessful.
Comments	Returns the latest response string from the host.

GroupOverview()

Syntax	<code>GroupOverview([range])</code>
Parameters	
<i>[range]</i>	The range for articles you want to view. The format for <i>range</i> is first-last, where first is "" (an empty string) or positive number, and last is "", a positive number, or the token end.
Return Value	An array of objects if successful. Each object contains one article, and the properties <code>articleDate</code> , <code>articleLines</code> , <code>articleNumber</code> , <code>from</code> , <code>messageID</code> , <code>otherHeaders</code> , <code>references</code> , and <code>subject</code> . The method returns an exception if unsuccessful.
Comments	Returns an overview for the articles in range for the group specified in the <code>Group</code> property.

ListGroups()

Syntax	<code>ListGroups([startDate])</code>
Parameters	
<i>[startDate]</i>	The earliest creation date to search. Groups created before this date are not listed. If you do not specify a start date, all groups are listed. The format for <i>startDate</i> is YYMMDD HHMMSS.
Return Value	An array of objects if successful. Each object contains the following properties, <code>Canpost</code> , <code>lastArticle</code> , <code>firstArticle</code> , and <code>group</code> . The method returns an exception if unsuccessful.
Comments	Lists the newsgroups available on the host.

PostArticle()

Syntax	<code>PostArticle()</code>
Parameters	None
Return Value	Null if successful. The article is stored in the document property. An exception if unsuccessful
Comments	Posts the article to the host, attaching files using MIME as necessary. The article is constructed using the following properties and methods: <ul style="list-style-type: none"> Header Properties <ul style="list-style-type: none"> • <code>From</code> • <code>Subject</code> • <code>Organization</code> • <code>To</code> • <code>ReplyTo</code> • <code>References</code> • <code>MaxHeadersLength</code> Body Properties/Methods <ul style="list-style-type: none"> • <code>ArticleText()</code> • <code>AddAttachment()</code> • <code>DeleteAttachment()</code>

SendCommand()

Syntax	<code>SendCommand(<i>string</i>)</code>
Parameters	
<i>string</i>	The string you are sending to the host.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Sends a string to the host without modification. This method is useful for interacting directly with the host using non-standard or unsupported extensions.

WLNntp()

Syntax	<code>new WLNntp()</code>
Parameters	None
Return Value	A new wlnntp object.
Comments	Creates a new wlnntp object, used to interact with the server.
Example	<code>myNewNntpObject = new WLNntp()</code>

NNTTP Sample Code

```

// Agenda Initialization
function InitAgenda () {
    IncludeFile("wlNntp.js",WLExecuteScript)
}
//=====

//Body Of Agenda
InfoMessage("Speed: "+wlGlobals.ConnectionSpeed)
nntp=new WLNntp()
wlGlobals.Debug=1;
InfoMessage("before login")
nntp.UserName="UserID"
nntp.Password="TopSecret"
nntp.Connect("hostname")
//=====

//Test ListGoups
/*v = nntp.ListGroups();
InfoMessage(v.length);
for (var i = 0; i < v.length; i++)
{
    InfoMessage("canPost = "+v[i].canPost);
    InfoMessage("first article = "+v[i].firstArticle);
    InfoMessage("group = "+v[i].group);
    InfoMessage("last article = "+v[i].lastArticle);
}
*/
//=====

//Test GroupOverview
/*nntp.Group="alt.groupname";
v = nntp.GroupOverview();
InfoMessage(v.length);
for (var i = 0; i < v.length; i++)
{
    InfoMessage("article date = "+v[i].articleDate);
    InfoMessage("article lines = "+v[i].articleLines);
    InfoMessage("article number = "+v[i].articleNumber);
    InfoMessage("article size = "+v[i].articleSize);
    InfoMessage("from = "+v[i].from);
    InfoMessage("messageId = "+v[i].messageId);
    InfoMessage("other headers = "+v[i].otherHeaders);
    InfoMessage("references = "+v[i].references);
    InfoMessage("subject = "+v[i].subject);
}
*/
//=====

```

```
//Test GetArticleCount
//nntp.Group="alt.groupname";
//InfoMessage (nntp.GetArticleCount ());
nntp.Group="alt.groupname";
InfoMessage (nntp.GetArticleCount ());
//=====

//Test GetArticle
/*nntp.Group="alt.groupname";
nntp.Outfile="c:\\temp\\article.txt";
nntp.GetArticle(1);
InfoMessage (nntp.document);
*/
//=====

//Test post article
nntp.From="poster name";
nntp.Subject="nntp test posting";
nntp.Organization="OrgName";
nntp.To="control.cancel, alt.groupname";
nntp.ReplyTo="poster@organization.org";
nntp.References="<referenceID@server.organization.org>";
nntp.MaxHeadersLength=100;
nntp.ArticleText="hello world";
//id1 = nntp.AddAttachment
//      ("c:\\temp\\file1.txt", "file", WLNntp.ENC_7BIT);
//id2 = nntp.AddAttachment
//      ("c:\\temp\\file2.txt", "file", WLNntp.ENC_7BIT);
//id5 = nntp.AddAttachment
//      ("c:\\downloaded.gif", "file", WLNntp.ENC_BASE64);
//id3 = nntp.AddAttachment
//      ("c:\\temp\\file3.txt", "file", WLNntp.ENC_7BIT);
//id4 = nntp.AddAttachment
//      ("c:\\temp\\file4.txt", "file", WLNntp.ENC_7BIT);
//nntp.DeleteAttachment(id3);
//nntp.DeleteAttachment(id1);
//nntp.DeleteAttachment(id4);
try          //catch to handle exceptions
{
    nntp.PostArticle();
}
catch (e)
{
    InfoMessage ("Error" + e)
}
//=====
```

```
//InfoMessage (nntp.GetStatusLine ());  
nntp.Disconnect ()  
delete nntp  
InfoMessage ("done")
```

wIPOP Object

The `wlPOP` object provides support for POP3 (Post Office Protocol) load and functional testing within WebLOAD. Support for standard POP operation is included. POP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect ()` method; otherwise an exception is thrown.

To access the `wlPOP` object, you must include the `wlPop.js` file in your `InitAgenda ()` function.

wIPOP Properties

AutoDelete

The `AutoDelete` property lets you specify whether or not to automatically delete an email once it has been read. You use this property to save or remove messages from your host. For example:

```
pop.AutoDelete = status
```

document

The `document` property is an object with four properties:

- ◆ `Headers` - A string containing the header of the message
- ◆ `MessageText` - A string containing the text of the message
- ◆ `Size` - An integer describing the size of the message in bytes
- ◆ `Attachments` - An array of objects, with each attachment existing as an object with the following properties:
 - `contentencoding` - The encoding of the attachment
 - `contenttype` - The content type of the attachment
 - `filename` - The file name of the attachment
 - `messagetext` - The text of the attachment
 - `partname` - The part name of the message

- `size` - The size of the attachment in bytes

For example:

```
var recentdocument = pop.document
var messageheaders = recentdocument.MessageHeaders
var messagetext = recentdocument.MessageText
var messagesize = recentdocument.MessageSize
var messageattachments = recentdocument.attachments
```

Headers[]

The `Headers` property is an array of objects containing header information from the host. Each object contains a key and an array of headers. For example:

```
var headersvalue = pop.Headers[0]
var headerskey=headersvalue.key
var headerstringvalues=headersvalue.values[0]
```

MaxLines

The `MaxLines` property lets you specify the maximum number of lines per email to retrieve from a POP host. You use this property to specify the number of lines to retrieve from each email. For example:

```
pop.Maxlines = numberoflines
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save a file or message locally on your computer. When you write to the `Outfile`, you overwrite the existing content. To avoid overwriting the existing content, you must specify a new `Outfile` each time you write. For example:

```
pop.Outfile = filename
```

Password

The `Password` property lets you specify a password when logging on to a host. You use this property to log onto a restricted POP host. WebLOAD automatically sends the password to the POP host when a wIPOP object connects to an POP host. For example:

```
pop.Password = password
```

Caution: The password appears in plain text in the Agenda. The password is visible to any user who has access to the Agenda

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = pop.Size
```

UserName

The `UserName` property lets you specify a User ID when logging on to a host. You use this property to log onto a restricted POP host. WebLOAD automatically sends the user name to the POP host when a `wlPOP` object connects to an POP host. For example:

```
pop.UserName = username
```

wlSource

The `wlSource` property contains the encoded multipart source of the message. This is the format in which the message is stored in the `Outfile` property. For example:

```
var messagesource = pop.wlSource
```

wlPOP Methods

Connect()

Syntax	<code>Connect(host, [port])</code>
Parameters	
<code>host</code>	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
<code>[port]</code>	The port to which you are connecting. If you do not specify a port, the default POP port is used.
Return Value	An exception if unsuccessful. On success the return value is undefined.
Comments	Starts a POP session with the host. When you connect, you are connecting to a specific mailbox within the host, as specified by your UserID.

Delete()

Syntax	<code>Delete ([MessageID])</code>
Parameters	
<i>messageID</i>	The identifier of the message you want to delete. If you do not specify a message ID, the current message is deleted.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the message with the corresponding ID. If no ID is specified, then the current message is deleted.

Disconnect()

Syntax	<code>Disconnect ()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the POP server.

GetCurrentMessageID()

Syntax	<code>GetCurrentMessageID ()</code>
Parameters	None
Return Value	The ID of the current message if successful, an exception if unsuccessful.
Comments	Returns the ID of the current message.

GetMailboxSize()

Syntax	<code>GetMailboxSize ()</code>
Parameters	None
Return Value	A string describing the size of the mailbox in bytes if successful.
Comments	Returns the total size of the mailbox in bytes.

GetMessageCount()

Syntax	<code>GetMessageCount ()</code>
Parameters	None
Return Value	A string containing the number of messages on the host if successful.
Comments	Returns the number of messages waiting on the host.

GetStatusLine()

Syntax	<code>GetStatusLine ()</code>
Parameters	None
Return Value	A string containing the latest response string if successful, an exception if unsuccessful.
Comments	Returns the latest response string from the host.

Reset()

Syntax	<code>Reset ()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Undoes all actions, including deletions, returning the host to its state at the start of the session. If this call is not made, disconnecting from the POP host applies all actions.

Retrieve()

Syntax	<code>Retrieve ([MessageID])</code>
Parameters	
<i>MessageID</i>	The identifier of the message you want to retrieve. If you do not specify a message ID, the next message is returned.
Return Value	Returns the message and populates the document property
Comments	Returns the message with the corresponding ID. If no ID is specified, then the next message is returned

SendCommand()

Syntax	SendCommand(<i>string</i>)
Parameters	
<i>string</i>	The string you are sending to the host.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Sends a string to the host without modification. This method is useful for interacting directly with the host using non-standard or unsupported extensions.

WLPop()

Syntax	new WLPop()
Parameters	None
Return Value	A new wIPOP object.
Comments	Creates a new wIPOP object, used to interact with the server.
Example	var myNewPopObject = new WLPop();

POP Sample Code

```
// Agenda Initialization
function InitAgenda() {
    IncludeFile("wIPOP.js",WLExecuteScript)
}
/*function InitClient() {

}*/
/*function TerminateClient() {
    delete pop;
}*/
//=====

//Body Of Agenda.
//InfoMessage("Speed: "+wIGlobals.ConnectionSpeed)
wIGlobals.Debug=1
var pop=new WLPop();
pop.UserName="UserID"
pop.PassWord="TopSecret"
pop.Connect("00.0.0.00");
//=====
```

```

//Test General Functions
/*count = pop.GetMessageCount();
InfoMessage("number of messages= "+ count);
count = pop.GetMailboxSize();
InfoMessage("size= "+ count);
status = pop.GetStatusLine();
InfoMessage("status= "+ status);
pop.SendCommand("hello");
status = pop.GetStatusLine();
InfoMessage("status= "+ status);
*/
//=====

//Test Delete And Reset
//two tests:
//1. if run as is, # of msgs should remain the same
//2. if run with pop.Reset commented out, # of msgs should be smaller
InfoMessage("number of messages= "+ pop.GetMessageCount());
//InfoMessage(pop.GetCurrentMessageID);
//pop.MaxLines=0;
pop.Delete(15);
InfoMessage("number of messages= "+ pop.GetMessageCount());
//InfoMessage(pop.GetCurrentMessageID);
//pop.Reset();
pop.Disconnect();
pop.Connect("00.0.0.00");
InfoMessage(pop.GetStatusLine());
//InfoMessage(pop.GetCurrentMessageID);
InfoMessage("number of messages= "+ pop.GetMessageCount());
//=====

//Test Retrieve
//InfoMessage("number of messages= "+ pop.GetMessageCount());
//InfoMessage(pop.GetCurrentMessageID);
//pop.AutoDelete=true
/*pop.Outfile="*.xyz";
//pop.MaxLines=0;
var count = pop.GetMessageCount();
InfoMessage(count);
for(var w = 1; w <= count; w++)
{
    pop.Retrieve(w);
    InfoMessage(pop.document.headers);
    InfoMessage(pop.document.messageText);
    InfoMessage(pop.document.size);
    InfoMessage(pop.document.attachments.length);
    for (var j = 0; j < pop.document.attachments.length; j++)
    {
        InfoMessage
            (pop.document.attachments[j].contentEncoding);
        InfoMessage(pop.document.attachments[j].contentType);
    }
}

```

```
        InfoMessage (pop.document.attachments[j].filename);
        InfoMessage (pop.document.attachments[j].messageText);
        InfoMessage (pop.document.attachments[j].partName);
        InfoMessage (pop.document.attachments[j].size);
    }
    InfoMessage ("Headers:");
    for (var i = 0; i < pop.Headers.length; i++)
    {
        for (var j = 0; j < pop.Headers[i].values.length; j++)
        {
            InfoMessage (pop.Headers[i].key + " = " +
                pop.Headers[i].values[j]);
        }
    }
    InfoMessage ("body"+pop.wISource);
    }*/
catch (e)
{
    InfoMessage ("Error" + e)
}
pop.Disconnect();
//=====
```

wSMTP Object

The `wSMTP` object provides support for SMTP (Mail Transfer Protocol) load and functional testing within WebLOAD. Support for standard SMTP operation is included. SMTP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect ()` method; otherwise an exception should be thrown.

To access the `wSMTP` object, you must include the `wSmtplib.js` file in your `InitAgenda ()` function.

wSMTP Properties

Attachments

The `Attachments` property lets you specify an attachment to an email message. The `filename` parameter is the name of the local file or datastream that you want to attach to the email message. For example:

```
smtp.Attachments = filename
```

AttachmentsEncoding

The `AttachmentsEncoding` property lets you specify the type of encoding you are applying to an email attachment. This property must be specified for each attachment. Valid values are:

- ◆ `7Bit`
- ◆ `Quoted`
- ◆ `Base64`
- ◆ `8Bit`
- ◆ `8BitBinary`

You may also specify the encoding using the following constants:

- ◆ `WLSmtplib.ENC_7BIT` - 7bit encoding
- ◆ `WLSmtplib.ENC_QUOTED` - Quoted Printable encoding

- ◆ `WLSmtp.ENC_BASE64` - Base64 encoding
- ◆ `WLSmtp.ENC_8BIT` - 8Bit encoding
- ◆ `WLSmtp.ENC_8BITBINARY` - Binary encoding

For example:

```
smtp.AttachmentsEncoding = encodingtype
```

AttachmentsTypes

The `AttachmentsTypes` property lets you specify the type of attachment you are including in an email message. This property must be specified for each attachment. Valid values are:

- ◆ `True` - specifies a type of file
- ◆ `False` - specifies a type of data

For example:

```
smtp.AttachmentsTypes = typeofattachment
```

Bcc

The `Bcc` property lets you specify the email addresses of additional recipients to be blind copied in an email. You may specify multiple addresses in a semicolon-separated list. You must specify this property with every email. Addresses may be specified in the format of "Me@MyCompany.com" or as "My Name <Me@MyCompany.com>". For example:

```
smtp.Bcc = blindcopyaddresses
```

Cc

The `Cc` property lets you specify the email addresses of additional recipients to be copied in an email. You may specify multiple addresses in a semicolon-separated list. You must specify this property with every email. Addresses may be specified in the format of "Me@MyCompany.com" or as "My Name <Me@MyCompany.com>". For example:

```
smtp.Cc = copyaddress; copyaddress
```

From

The `From` property lets you describe the Reply To in plain language. You may use this property to identify your Reply To email address in a plain language format. For example:

```
smtp.From = replyname
```

Message

The `Message` property lets you specify the text appearing in the body of your email. You use this property to write the text of the email message itself.

ReplyTo

The `ReplyTo` property lets you specify the return address of your email. You may specify multiple addresses in a semicolon-separated list. Addresses may be specified in the format of "Me@MyCompany.com" or as "My Name <Me@MyCompany.com>". For example:

```
smtp.ReplyTo = replyaddress
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = smtp.Size
```

Subject

The `Subject` property lets you specify the text appearing the subject field of your email. You use this property to provide a brief description of the contents of your email. For example:

```
smtp.Subject = subjectheader
```

To

The `To` property lets you specify a recipient's email address. You may specify multiple addresses in a semicolon-separated list. You must specify this property with every email. Addresses may be specified in the format of "Me@MyCompany.com" or as "My Name <Me@MyCompany.com>". For example:

```
smtp.To = recipientaddress; recipientaddress
```

Type

The `Type` property lets you specify the type of server with which you are working. The default value for this property is SMTP. Valid values are:

- ◆ SMTP - a standard SMTP server

◆ ESMTP - an extended SMTP server

For example:

```
smtp.Type = servertype
```

wSMTP Methods

AddAttachment()

Syntax	<code>AddAttachment(<i>string</i>, <i>type</i>, [<i>encoding</i>])</code>
Parameters	
<i>String</i>	The string you are sending to the host. If you are sending a file, the string is the location and name of the file. If you are sending a data attachment, the string is the data to be attached.
<i>Type</i>	The type of attachment you are sending. The default value is File. Valid values are: <ul style="list-style-type: none"> • File • Data
<i>encoding</i>	The type of encoding to apply to the file. The default value is 7Bit. Valid values are: <ul style="list-style-type: none"> • 7Bit • Quoted • Base64 • 8Bit • 8BitBinary
Return Value	Returns an integer value Attachment ID if successful, an exception if unsuccessful.
Comments	Adds an attachment to the email message.

Connect()

Syntax	<code>Connect(<i>host</i>, [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.

<i>port</i>	The port to which you are connecting. If you do not specify a port, the default SMTP port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts an SMTP session with the host.

DeleteAttachment()

Syntax	DeleteAttachment (<i>ID</i>)
Parameters	
<i>ID</i>	The ID of the attachment you are deleting.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Removes an attachment from the email message.

Disconnect()

Syntax	Disconnect ()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the SMTP host.

Send()

Syntax	Send ()
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Sends mail to recipients, attaching files using MIME as necessary. After sending the attachments, data is deleted.

SendCommand()

Syntax	SendCommand (<i>string</i>)
Parameters	
<i>string</i>	The string you are sending to the host.
Return Value	Null if successful, an exception if unsuccessful.

Comments Sends a string to the host without modification. This method is useful for interacting directly with the host using non-standard or unsupported extensions.

Verify()

Syntax Verify()

Parameters None

Return Value Returns a 1 if the address is valid, a 0 if the address is invalid. If the method is unable to verify the address due to authentication or other reasons, it returns an exception.

Comments Checks that the address in the `To` property is valid. To use this method, include only one address in the `To` property.

WLSmtp()

Syntax new WLSmtp()

Parameters None

Return Value A new wSMTP object.

Comments Creates a new wSMTP object, used to interact with the server.

Example

```
function InitClient() {
    myNewSmtpObject = new WLSmtp()
}
```

SMTP Sample Code

```
// Agenda Initialization
function InitAgenda() {
    IncludeFile("wLSmtp.js",WLExecuteScript)
    // include the file that enables SMTP
}

function InitClient() {
    Smtp=new WLSmtp() // create the new SMTP object
    Smtp.Connect("HostName"); // connect to the server
}

function TerminateClient() {
    Smtp.Disconnect(); // logout from the server
    delete Smtp // delete the SMTP object
}
//=====
```

```


// Body Of Agenda

//Test Send Attachments
Smtp.To=" \"Recipient Name\" <Recipient@recipient.com>";
Smtp.From= "Sender@sender.com";
Smtp.Cc="Copy1@copy.here.org, Copy2@copy.there.org";
                                                    // multiple CC's
Smtp.ReplyTo="Sender@sender.com";
                                                    // optional different reply to address
Smtp.Subject="Message Subject ";                // Text string
Smtp.Message="Greetings from the wLSMTP class"; // Message text

// Add attachments from local file using different
// encoding techniques
// 7BIT are text files, the BASE64 is for a binary file
// (in this case an image)
id1 = Smtp.AddAttachment
        ("c:\\file1.txt", "file", WLSmtp.ENC_7BIT);
id2 = Smtp.AddAttachment
        ("c:\\file2.txt", "file", WLSmtp.ENC_7BIT);
id3 = Smtp.AddAttachment
        ("c:\\file3.txt", "file", WLSmtp.ENC_7BIT);
id4 = Smtp.AddAttachment
        ("c:\\file4.txt", "file", WLSmtp.ENC_7BIT);
id5 = Smtp.AddAttachment
        ("c:\\downloaded.gif", "file", WLSmtp.ENC_BASE64);

// You may delete attachments prior to sending the mail message
Smtp.DeleteAttachment(id3);
Smtp.DeleteAttachment(id1);
Smtp.DeleteAttachment(id4);
Smtp.Send(); // and send it!

InfoMessage(Smtp.GetStatusLine());
            // print out the last response from the server

 ch (e)
    InfoMessage ("Error" + e)
    }
//=====
InfoMessage("done") // End of SMTP sample script

```

wITCP Object

The `wITCP` object provides support for TCP (Transfer Control Protocol) load and functional testing within WebLOAD. Support for standard TCP operation is included. TCP over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect()` method; otherwise an exception is thrown.

To access the `wITCP` object, you must include the `wITcp.js` file in your `InitAgenda()` function.

wITCP Properties

document

The `document` property contains all responses from the host since the last time the `Send()` method was used. Each time a message is returned, it is concatenated to the `document` object. The `document` may be cleared manually using the `Erase()` method. For example:

```
var recentdocument = tcp.document
```

InBufferSize

The `InBufferSize` property specifies the size, in bytes, of the incoming data buffer. To remove this setting, either delete the property, or set it to a negative value. For example:

```
tcp.InBufferSize = maximuminsize
```

LocalPort

The `LocalPort` property specifies the TCP port to which you are connecting. If you do not specify the `LocalPort` property, you connect to a randomly selected port. For example:

```
tcp.LocalPort = portnumber
```

NextPrompt

The `NextPrompt` property specifies the text for the Agenda to look for in the next prompt from the host. A `Receive()` call is viewed as successful if the prompt contains the text string specified by the `NextPrompt` variable. To specify a prompt with no message, specify a `NextPrompt` with an empty value, or delete the `NextPrompt` property. Once this property is specified, it limits all subsequent instances of the `Receive()` method. Either delete the property or set it to zero to remove the limitation. For example:

```
tcp.NextPrompt = promptmessage
```

NextSize

The `NextSize` property specifies the size, in bytes, of the expected data. If you specify a `NextSize` of 100 bytes, for example, the `Receive()` method returns to the Agenda when the document object contains 100 bytes of data. Once this property is specified, it limits all subsequent instances of the `Receive()` method. Either delete the property or set it to zero to remove the limitation. For example:

```
tcp.NextSize = expectedsize
```

OutBufferSize

The `OutBufferSize` property specifies the size, in bytes, of the outgoing data buffer. To remove this setting, either delete the property, or set it to a negative value. For example:

```
tcp.OutBufferSize = maximumoutsize
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save the responses from the host locally on your computer. You must specify the output file before calling the `Receive()` method to save the responses to that file.

You write to the output file each time you use the `Receive()` method. If you call the `Receive()` method more than once, you must specify a different output file each time, or you overwrite the previous output file. For example:

```
tcp.Outfile = filename
```

ReceiveMessageText

The `ReceiveMessageText` property returns the reason why the host stopped responding. You use this property to determine the state of the host. Possible values are:

- ◆ Prompt was found - The host returned the prompt specified in the `NextPrompt` property.
- ◆ Timeout - The last command exceeded the time limit specified by the `Timeout` property.
- ◆ Byte length reached - The host received the amount of data specified in the `NextSize` property.

```
InfoMessage (TCP.ReceiveMessageText) ;
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = tcp.Size
```

Timeout

The `Timeout` property lets you specify the length of the delay, in milliseconds, before the Agenda breaks its connection with the host. If you do not specify the timeout property, the Agenda may freeze if the host does not respond as you expect it to. To set an unlimited timeout, specify a value of zero, or a negative value. For example:

```
tcp.Timeout = timedelay
```

Note: It is recommended that you include a `Timeout` property in all Agendas that use the `wITCP` object. If you do not, and the Agenda fails to return a prompt, your session may freeze.

wITCP Methods

Connect()

Syntax	<code>Connect(<i>host</i>, [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
<i>port</i>	The port to which you are connecting. If you do not specify a port, the default TCP port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts a TCP session with the host.

Disconnect()

Syntax	<code>Disconnect()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the TCP host.

Erase()

Syntax	<code>Erase()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Clears the contents of the document object.

Receive()

Syntax	<code>Receive()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Returns all responses from the host since the last time the <code>Send()</code> method was used. A <code>Receive()</code> method returns to the Agenda when the <code>NextPrompt</code> , <code>NextSize</code> , or <code>Timeout</code> properties are met. If more than one of these properties is specified, the method returns to the Agenda when the first one is met. Subsequent uses of <code>Receive()</code> find the next instance of the limiting property, returning additional information from the buffer. The content returned depends upon which of the three limiting properties triggered the return.

Send()

Syntax	<code>Send(data_to_send)</code>
Parameters	
<i>data_to_send</i>	The data that you want to send to the host.
Return Value	A string containing the response from the host if successful, an exception if unsuccessful.
Comments	Sends data to the host via TCP and clears the document object.

WLTcp()

Syntax	<code>new WLTcp()</code>
Parameters	None
Return Value	A new wTCP object.
Comments	Creates a new wTCP object, used to interact with the server.
Example	<pre>function InitClient() { myNewTcpObject = new WLTcp(); }</pre>

TCP Sample Code

```
// Agenda Initialization
function InitAgenda() {
    IncludeFile("wlTcp.js",WLExecuteScript)
}

function InitClient() {
    tcp=new WLTcp();
}

function TerminateClient()
{
    delete tcp;
}
//=====

//Body Of Agenda.
InfoMessage("Speed: "+wlGlobals.ConnectionSpeed)
wlGlobals.Debug=1;
tcp.Outfile = "c:\\tcp.txt";
tcp.Timeout = 2000;
tcp.NextPrompt = "\r\n\r\n";
//tcp.NextSize=1900;
//=====

try
{
    tcp.Connect("www.sitename.com", 80);
    tcp.Send("GET /products/index.htm HTTP/1.0\r\n\r\n");
    //Sleep(3000);
    tcp.Receive();
    InfoMessage(tcp.document);
    InfoMessage(tcp.ReceiveMessageText);
    tcp.NextSize=10091;
    tcp.NextPrompt="";
    tcp.Erase();
    tcp.Receive();
    InfoMessage(tcp.document);
    InfoMessage(tcp.ReceiveMessageText);
}

catch(e)
{
    InfoMessage(e);
}
//=====
InfoMessage("done");
```

wITelnet Object

The `wITelnet` object provides support for Telnet load and functional testing within WebLOAD. Support for standard Telnet operation is included. Telnet over secure connections (SSL) is not currently supported.

If a connection is required but has expired or has not yet been established, the underlying code attempts to login. Logging in requires you to call the appropriate `Connect()` method otherwise an exception is thrown.

To access the `wITelnet` object, you must include the `wITelnet.js` file in your `InitAgenda()` function.

wITelnet Properties

document

The `document` property contains all responses from the host since the last time the `Send()` method was used. Each time a message is returned, it is concatenated to the `document` object. The `document` may be cleared manually using the `Erase()` method. For example:

```
var recentdocument = telnet.document
```

NextPrompt

The `NextPrompt` property specifies the text for the Agenda to look for in the next prompt from the host. A `Receive()` call is viewed as successful if the prompt contains the text string specified by the `NextPrompt` variable. To specify a prompt with no message, specify a `NextPrompt` with an empty value, or delete the `NextPrompt` property. Once this property is specified, it limits all subsequent instances of the `Receive()` method. Either delete the property or set it to zero to remove the limitation. For example:

```
telnet.NextPrompt = promptmessage
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save the responses from the host locally on your computer. You must specify the output file before calling the `Receive()` method to save the responses to that file.

You write to the output file each time you use the `Receive()` method. If you call the `Receive()` method more than once, you must specify a different output file each time, or you overwrite the previous output file. For example:

```
telnet.Outfile = filename
```

ReceiveMessageText

The `ReceiveMessageText` property returns the reason why the host stopped responding. You use this property to determine the state of the host. Possible values are:

- ◆ Prompt was found - The host returned the prompt specified in the `NextPrompt` property.
- ◆ Timeout - The last command exceeded the time limit specified by the `Timeout` property.
- ◆ Byte length reached - The host received the amount of data specified in the `NextSize` property.

For example:

```
InfoMessage(Telnet.ReceiveMessageText);
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = telnet.Size
```

Timeout

The `Timeout` property lets you specify the length of the delay, in milliseconds, before the Agenda breaks its connection with the host. If you do not specify the timeout property, the Agenda may freeze if the host does not respond as you expect it to. To set an unlimited timeout, specify a value of zero, or a negative value. For example:

```
telnet.Timeout = timedelay
```

Note: It is recommended that you include a `Timeout` property in all Agendas that use the `wlTelnet` object. If you do not, and the Agenda fails to return a prompt, your session may freeze.

wITelnet Methods

Connect()

Syntax	<code>Connect(<i>host</i>, [<i>port</i>])</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
<i>port</i>	The port to which you are connecting. If you do not specify a port, the default Telnet port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts a Telnet session with the host.

Disconnect()

Syntax	<code>Disconnect()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful
Comments	Terminates a connection to the Telnet host.

Erase()

Syntax	<code>Erase()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful
Comments	Clears the contents of the document object.

Receive()

Syntax	<code>Receive()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.

Comments	Returns all responses from the host since the last time the <code>Send()</code> method was used. A <code>Receive()</code> method returns to the Agenda when the <code>NextPrompt</code> , <code>NextSize</code> , or <code>Timeout</code> properties are met. If more than one of these properties is specified, the method returns to the Agenda when the first one is met. Subsequent uses of <code>Receive()</code> find the next instance of the limiting property, returning additional information from the buffer. The content returned depends upon which of the three limiting properties triggered the return.
-----------------	--

Send()

Syntax	<code>Send(data_to_send)</code>
Parameters	
<i>data_to_send</i>	The data that you want to send to the host.
Return Value	A string containing the response from the host if successful, an exception if unsuccessful.
Comments	Sends data to the host via Telnet and clears the document object.

WLTelnet()

Syntax	<code>new WLTelnet()</code>
Parameters	None
Return Value	A new <code>wLTelnet</code> object.
Comments	Creates a new <code>wLTelnet</code> object, used to interact with the server.
Example	<pre>function InitClient() { myNewTelnetObject = new WLTelnet() }</pre>

Telnet Sample Code

```
// Agenda Initialization
function InitAgenda() {
    IncludeFile("wLTelnet.js",WLExecuteScript)
    // include the file that enables Telnet
}

function InitClient() {
    Telnet=new WLTelnet()           // create a new telnet object
}

function TerminateClient()
{
```



```

    delete Telnet          // delete the object we were using
}
//=====

// Body Of Agenda
// Set timeout and prompt
// IMPORTANT: Set a timeout when setting a prompt. Otherwise,
// If the prompt is unexpected or incorrect the Agenda will
// freeze while waiting for a prompt that will never arrive
Telnet.Timeout=1000;      // one second
Telnet.NextPrompt="User name: ";      // text to look for

Telnet.Connect("000.0.0.0");      // connect
Telnet.Receive();      // wait for data from the remote host
Telnet.Send("myname");      // send login name
InfoMessage(Telnet.document);      // write out the data received
InfoMessage(Telnet.ReceiveMessageText);
// write out why the call returned
Telnet.NextPrompt="Password: ";      // next prompt to look for
Telnet.Receive();      // wait for data
Telnet.Outfile="c:\\filename.txt";
// save this next response to file as well

InfoMessage(Telnet.document);      // what did we get?
InfoMessage(Telnet.ReceiveMessageText);
// write out why the call returned
Telnet.Send("mypassword");      // send password
Telnet.NextPrompt=">";      // new prompt to wait for
Telnet.Receive();      // wait for a response

Telnet.Send("command");      // send command text to the host
Telnet.Receive();      // wait for a response
InfoMessage(Telnet.document);      // what did we get?
InfoMessage(Telnet.ReceiveMessageText);
// write out why the call returned
Telnet.Disconnect();      // finally disconnect
//=====

//This is another way to work with telnet. When no prompt
//is set the timeout is ignored. Instead the agenda writer
//must manually keep receiving the data by calling the receive
//command. Receive() returns the response as well as assigning
//the value to the this.document property. It is up to the user
//to perform a delay before he/she receives the data.

Telnet.Connect("000.0.0.0");      // log in to a remote host
// In this case we receive three times.
// In your script you may keep calling Receive() until the
// telnet object's document property contains the data you are
// looking for, or until you decide to do something else

```

```
Telnet.Receive(); // fetch the data
Telnet.Receive(); // Wait for more
Telnet.Receive(); // Wait for more
InfoMessage(Telnet.document); // Contains text from ALL receives
InfoMessage(Telnet.ReceiveMessageText); // reason calls returned
Telnet.Send("Command"); // clears the document object

Telnet.Receive(); // fetch the data
Telnet.Receive(); // Wait for more
Telnet.Receive(); // Wait for more
InfoMessage(Telnet.document);
InfoMessage(Telnet.ReceiveMessageText);

Telnet.Send("command");

Telnet.Receive();
Telnet.Receive(); // Wait for more
Telnet.Receive(); // Wait for more
InfoMessage(Telnet.document);
InfoMessage(Telnet.ReceiveMessageText);

Telnet.Send("dir");

Telnet.Receive();
Telnet.Receive(); // Wait for more
Telnet.Receive(); // Wait for more
InfoMessage(Telnet.document);
InfoMessage(Telnet.ReceiveMessageText);

catch (e)
{
    InfoMessage ("Error" + e)
}

Telnet.Disconnect(); // log out from the remote host
InfoMessage("done") // End of telnet sample script
```

wIUDP Object

The `wIUDP` object provides support for UDP (User Datagram Protocol) load and functional testing within WebLOAD. Support for standard UDP operation is included. UDP over secure connections (SSL) is not currently supported.

To access the `wIUDP` object, you must include the `wlUdp.js` file in your `InitAgenda()` function.

wIUDP Properties

document

The `document` property is an array of objects sent in the current session, with each object containing the following properties:

- ◆ `datagram` - The datagram retrieved from the database
- ◆ `address` - The address of the datagram
- ◆ `port` - The port used to communicate with the database

The `document` property contains all responses from the host since the last time the `Send()` method was used. Each time a message is returned, it is concatenated to the `document` object. The `document` may be cleared manually using the `Erase()` method. For example:

```
var recentdocument = udp.document
```

InBufferSize

The `InBufferSize` property specifies the size, in bytes, of the incoming data buffer. For example:

```
udp.InBufferSize = maximuminsize
```

LocalHost

The `LocalHost` property lets you specify a local host for use in broadcasting via UDP. For example:

```
udp.LocalHost = localhostname
```

LocalPort

The `LocalPort` property specifies the UDP port to which you are connecting. If you do not specify the `LocalPort` property, you connect to a randomly selected port. For example:

```
udp.LocalPort = portnumber
```

MaxDatagramSize

The `MaxDatagramSize` property specifies the maximum size, in bytes, of datagrams that you may send or receive via UDP. For example:

```
udp.MaxDatagramSize = maximumsize
```

NumOfResponses

The `NumOfResponses` property specifies the number of responses the testing machine waits for before proceeding. You use this property to make sure that all of your hosts have responded. To specify an unlimited number of responses, specify a `NumOfResponses` value of zero. For example:

```
udp.NumOfResponses = numberofhosts
```

OutBufferSize

The `OutBufferSize` property specifies the size, in bytes, of the outgoing data buffer. For example:

```
udp.OutBufferSize = maximumoutsize
```

Outfile

The `Outfile` property lets you specify the name of an output file. You use this property to save the responses from the host locally on your computer. You must specify the output file before calling the `Receive()` method to save the responses to that file.

You write to the output file each time you use the `Receive()` method. If you call the `Receive()` method more than once, you must specify a different output file each time, or you will overwrite the previous output file. For example:

```
udp.Outfile = filename
```

ReceiveMessageText

The `ReceiveMessageText` property returns the reason why the host stopped responding. You use this property to determine the state of the host. Possible values are:

- ◆ Prompt received - The host returned a prompt and is waiting for further instructions.
- ◆ Timeout - The last command exceeded the limit specified by the `Timeout` property.
- ◆ No prompt specified - The host is unable to return a prompt. Often, this means there is an error in the Agenda.

For example:

```
InfoMessage (udp.ReceiveMessageText) ;
```

RequestedPackets

The `RequestedPackets` property specifies the number of packets the testing machine waits for before proceeding. To specify an unlimited number of packets, specify a `RequestedPackets` value of zero. For example:

```
udp.RequestedPackets = numberofpackets
```

Size

The `Size` property returns the byte length of data transferred to the host. You use this property to compare starting and finishing sizes to verify that files have arrived without corruption. For example:

```
var filesize = udp.Size
```

Timeout

The `Timeout` property lets you specify the length of the delay, in milliseconds, before the Agenda breaks its connection with the host. If you do not specify the timeout property, the Agenda may freeze if the host does not respond as you expect it to. For example:

```
udp.Timeout = timedelay
```

Note: It is recommended that you include a `Timeout` property in all Agendas that use the wLUDP object. If you do not, and the Agenda fails to return a prompt, your session may freeze.

wIUDP Methods

Bind()

Syntax	<code>Bind()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Creates a UDP port and sets the <code>OutBufferSize</code> , <code>InBufferSize</code> , <code>MaxDatagramSize</code> , <code>LocalHost</code> , and <code>LocalPort</code> properties. The value of these properties is fixed when the <code>Bind()</code> method is used. To change the value of any of these properties, you must use the <code>UnBind()</code> method, change the value of the property and using the <code>Bind()</code> method again.

Broadcast()

Syntax	<code>Broadcast(port, data_to_send)</code>
Parameters	
<i>port</i>	The port to which you are connecting.
<i>data_to_send</i>	The data that you want to send to the local net.
Return Value	A string containing the response from the host if successful, an exception if unsuccessful.
Comments	Broadcasts data to the local net.

Erase()

Syntax	<code>Erase()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful
Comments	Clears the contents of the document property, setting it to an empty array.

Receive()

Syntax	<code>Receive()</code>
Parameters	None

Return Value	Null if successful, an exception if unsuccessful.
Comments	Returns all responses from the host since the last time the <code>Send()</code> method was used. The <code>Receive()</code> method returns to the Agenda when the <code>RequestedPackets</code> or <code>Timeout</code> property is met. Subsequent uses of <code>Receive()</code> find the next instance of the limiting property, returning additional information from the buffer.

Send()

Syntax	<code>Send(host, port, data_to_send)</code>
Parameters	
<i>host</i>	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
<i>port</i>	The port to which you are connecting.
<i>data_to_send</i>	The data that you want to send to the host.
Return Value	A string containing the response from the host if successful, an exception if unsuccessful.
Comments	Sends data to the host via UDP.

UnBind()

Syntax	<code>UnBind()</code>
Parameters	None
Return Value	Null if successful, an exception if unsuccessful.
Comments	Closes a UDP socket. You must use this command to close an existing UDP socket before you may use the <code>Bind()</code> again.

WLUDP()

Syntax	<code>new WLUDP()</code>
Parameters	None
Return Value	A new wLUDP object.
Comments	Creates a new wLUDP object, used to interact with the server.
Example	<pre>function InitClient() { myNewUDPObject = new WLUDP() }</pre>

UDP Sample Code

```

// Agenda Initialization
function InitAgenda() {
    IncludeFile("wlUdp.js",WLExecuteScript)
                                     // enable the UDP objects
}

function InitClient() {
    udp=new WLUDP();                 // create a new UDP object
}

function TerminateClient() {
    delete udp                       // delete the UDP object
}
//=====

//Body Of Agenda.

//Test Send: set the buffer sizes appropriately for the data
try
{
udp.OutBufferSize=10;
udp.InBufferSize=12;
udp.MaxDatagramSize=10;
udp.Timeout=10000;                 // 10 second timeout
udp.NumOfResponses=1; // return after one remote machine responds
udp.Outfile="c:\\serialize.txt";   // file to save responses to
udp.Bind();
udp.Send("00.0.0.00", 7, "good morning");
                                     // send a datagram to one machine on port 7
udp.Receive();                     // wait for a response
InfoMessage(udp.ReceiveMessageText); // This is what happened

// show the properties of the response
// note that the udp.document object is an array
InfoMessage(udp.document[0].datagram); // get the response
InfoMessage(udp.document[0].address); // which machine responded?
InfoMessage(udp.document[0].port);    // the port

// now broadcast to seven machines
udp.NumOfResponses=7; // we expect seven machines to respond
udp.Outfile="c:\\serialize.txt"; // send the responses
udp.Broadcast(7, "good morning");
                                     // send the message (again on port 7)
udp.Receive();                     // wait for the responses
InfoMessage(udp.ReceiveMessageText); // print the return reason

// For each host that responded there will be an entry
// in the array. This loop examines each one
for (var i = 0; i < udp.document.length; i++)
{

```



```
        InfoMessage("datagram= "+udp.document[i].datagram);
        InfoMessage("address= "+udp.document[i].address);
        InfoMessage("port= "+udp.document[i].port);
    }
}
catch (e)
{
    InfoMessage ("Error" + e)
}
//=====
InfoMessage("done")           // end of the UDP sample script
```


HTTP Protocol Status Messages

This appendix documents the HTTP protocol status messages that you may see over the course of a typical test session. The status-code definitions provided in this appendix include a list of method(s) that the status code may follow and any meta information required in the response. The material included here is part of the HTTP protocol standard provided by the IETF.

The HTTP protocol status messages fall into the following categories:

- ◆ Informational (*1XX*)
- ◆ Success (*2XX*)
- ◆ Redirection (*3XX*)
- ◆ Client Error (*4XX*)
- ◆ Server Error (*5XX*)

D.1 Informational 1XX

The *1XX* class of status code indicates a provisional response, consisting only of the `Status-Line` and optional headers, and is terminated by an empty line. There are no required headers for this class of status code. Since HTTP/1.0 did not define any *1XX* status codes, servers *must not* send a *1XX* response to an HTTP/1.0 client except under experimental conditions.

A client must be prepared to accept one or more *1XX* status responses prior to a regular response, even if the client does not expect a `100 (Continue)` status message. Unexpected *1XX* status responses may be ignored by a user agent.

Proxies must forward *1XX* responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the *1XX* response. (For example, if a proxy adds an `Expect: 100-continue` field when it forwards a request, then it need not forward the corresponding `100 (Continue)` response(s).)

Table D-1: Informational 1XX message set

Message	Description
100 Continue	The client should continue with request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client should continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server must send a final response after the request has been completed.
101 Switching Protocols	<p>The client has requested, via the <code>Upgrade</code> message header field, a change in the application protocol being used on this connection. This response indicates that the server understands and is willing to comply with the client's request. The server will switch protocols to those defined by the response's <code>Upgrade</code> header field immediately after the empty line which terminates this <code>101</code> response.</p> <p>The protocol should be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features.</p>

D.2 Success 2XX

The 2XX class of status code indicates that the client's request was successfully received, understood, and accepted.

Table D-2: Successful 2XX message set

Message	Description
200 OK	<p>The request has succeeded. The information returned with a 200 response is dependent on the method used in the request. For example:</p> <ul style="list-style-type: none"> ▪ GET—an entity corresponding to the requested resource is sent in the response ▪ HEAD—the entity-header fields corresponding to the requested resource are sent in the response without any message-body ▪ POST—an entity describing or containing the result of the action ▪ TRACE—an entity containing the request message as received by the end server
201 _Created	<p>The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource identified by the <code>Location</code> header field.</p> <p>A 201 response should include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type identified in the <code>Content-Type</code> header field. The origin server must create the resource before returning a 201 status code. If the action cannot be carried out immediately, the server should respond with 202 (Accepted) response instead.</p> <p>A 201 response may contain an <code>ETag</code> response header field indicating the current value of the entity tag for the requested variant just created.</p>
202 Accepted	<p>The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, depending on whether or not it is authorized or disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.</p> <p>The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response should include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.</p>

Message	Description
203 Non-Authoritative Information	The metainformation being returned in the entity-header is not the definitive set that is usually obtained from the origin server. This information has been gathered from a local or a third-party copy. The set presented may be a subset or superset of the original version. For example, including local annotation information about the resource might result in a superset of the metainformation known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be a generic (perhaps non-informative) 200 (OK).
204 No Content	<p>The server has fulfilled the request, does not need to return an entity-body, and might want to return updated metainformation. The response may include new or updated metainformation in the form of entity-headers, which if present should be associated with the requested variant.</p> <p>If the client is a user agent, it <i>should not change its document view</i> from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation should be applied to the document currently in the user agent's active view.</p> <p>The 204 response must not include a message-body, and thus is always terminated by the first empty line after the header fields.</p>
205 Reset Content	The server has fulfilled the request and the user agent <i>should reset the document view</i> which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is entered so that the user can easily initiate another input action. The response must not include an entity.
206 Partial Content	<p>The server has fulfilled the partial GET request for the resource. The request must have included a Range header field indicating the desired range. The request may have also included an If-Range header field to make the request conditional.</p> <p>The response must include the following header fields:</p> <ul style="list-style-type: none"> ▪ Either a Content-Range header field indicating the range included with this response, or a multipart/byteranges Content-Type field including Content-Range fields for each part. If a Content-Length header field is present in the response, its value must match the actual number of OCTETs transmitted in the message-body. ▪ Date ▪ ETag and/or Content-Location, if the header would have been sent in a 200 response to the same request

Message	Description
	<ul style="list-style-type: none"><li data-bbox="678 394 1349 474">▪ Expires, Cache-Control, and/or Vary, if the field-value might differ from that sent in any previous response for the same variant. <p data-bbox="678 495 1383 716">If the 206 response is the result of an If-Range request that used a strong cache validator, the response should not include other entity-headers. If the response is the result of an If-Range request that used a weak validator, the response must not include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers. Otherwise, the response must include all of the entity-headers that would have been returned with a 200 (OK) response to the same request.</p> <p data-bbox="678 737 1383 789">A cache must not combine a 206 response with other previously cached content if the ETag or Last-Modified headers do not match exactly.</p> <p data-bbox="678 810 1383 863">A cache that does not support Range and Content-Range headers must not cache 206 (Partial) responses.</p>

D.3 Redirection 3XX

The *3XX* class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required may be carried out by the user agent without interaction with the user if and only if the method used in the second request is `GET` or `HEAD`. A client should detect infinite redirection loops, since such loops generate network traffic for each redirection.

Note: Previous versions of this specification recommended a maximum of five redirections. Content developers should be aware that there might be clients that implement such a fixed limitation.

Table D-3: Redirection 3XX message set

Message	Description
300 Multiple Choices	<p>The requested resource corresponds to any one of a set of representations, each with its own specific location. Agent-driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.</p> <p>Unless it was a <code>HEAD</code> request, the response should include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type identified in the <code>Content-Type</code> header field. Depending upon the format and the capabilities of the user agent, the most appropriate choice may be selected automatically. However, this specification does not define any standard for such automatic selection.</p> <p>If the server has a preferred choice of representation, it should include the specific URI for that representation in the <code>Location</code> field. User agents may use the <code>Location</code> field value for automatic redirection. This response is cacheable unless otherwise indicated.</p>
301 Moved Permanently	<p>The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the <code>Request-URI</code> to one or more of the new references returned by the server, where possible. This response is cacheable unless otherwise indicated.</p> <p>The new permanent URI should be identified by the <code>Location</code> field in the response. Unless the request method was <code>HEAD</code>, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s).</p> <p>If the <code>301</code> status code is received in response to a request other than <code>GET</code> or <code>HEAD</code>, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the</p>

Message	Description
	<p>conditions under which the request was issued.</p> <p>Note: When automatically redirecting a <code>POST</code> request after receiving a <code>301</code> status code, some existing HTTP/1.0 user agents will erroneously change it into a <code>GET</code> request.</p>
<p>302 Found</p>	<p>The requested resource <i>temporarily</i> resides under a different URI. Since the redirection might be altered on occasion, the client should continue to use the <code>Request-URI</code> for future requests. This response is only cacheable if indicated by a <code>Cache-Control</code> or <code>Expires</code> header field.</p> <p>The temporary URI should be identified by the <code>Location</code> field in the response. Unless the request method was <code>HEAD</code>, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s).</p> <p>If the <code>302</code> status code is received in response to a request other than <code>GET</code> or <code>HEAD</code>, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.</p> <p>Note: RFC 1945 and RFC 2068 specify that the client is not allowed to change the method on the redirected request. However, most existing user agent implementations treat <code>302</code> as if it were a <code>303</code> response, performing a <code>GET</code> on the <code>Location</code> field-value regardless of the original request method. The status codes <code>303</code> and <code>307</code> have been added for servers that wish to make unambiguously clear which kind of reaction is expected of the client.</p>
<p>303 See Other</p>	<p>The response to the request can be found under a different URI and should be retrieved using a <code>GET</code> method on that resource. This method exists primarily to allow the output of a <code>POST</code>-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource. The <code>303</code> response must not be cached, but the response to the second (redirected) request might be cacheable.</p> <p>The different URI should be identified by the <code>Location</code> field in the response. Unless the request method was <code>HEAD</code>, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s).</p> <p>Note: Many pre-HTTP/1.1 user agents do not understand the <code>303</code> status. When interoperability with such clients is a concern, the <code>302</code> status code may be used instead, since most user agents react to a <code>302</code> response as described here for <code>303</code>.</p>

Message	Description
304 Not Modified	<p>If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server should respond with this status code. The 304 response must not contain a message-body, and thus is always terminated by the first empty line after the header fields.</p> <p>The response must include the following header fields:</p> <ul style="list-style-type: none"> ▪ Date, unless its omission is required. ▪ If a clockless origin server obeys these rules, and proxies and clients add their own Date to any response received without one, (as already specified by [RFC 2068]), caches will operate correctly. ▪ ETag and/or Content-Location, if the header would have been sent in a 200 response to the same request. ▪ Expires, Cache-Control, and/or Vary, if the field-value might differ from that sent in any previous response for the same variant. <p>If the conditional GET used a strong cache validator, the response should not include other entity-headers. If the conditional GET used a weak validator, the response <i>must not</i> include other entity-headers. This prevents inconsistencies between cached entity-bodies and updated headers.</p> <p>If a 304 response indicates an entity not currently cached, then the cache must disregard the response and repeat the request without the conditional.</p> <p>If a cache uses a received 304 response to update a cache entry, the cache must update the entry to reflect any new field values given in the response.</p>
305 Use Proxy	<p>The requested resource must be accessed through the proxy identified by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses must only be generated by origin servers.</p> <p>Note: RFC 2068 did not clearly state that 305 was intended to redirect a single request, and to be generated by origin servers only. Nevertheless, not observing these limitations has significant security consequences.</p>
306 (Unused)	<p>The 306 status code was used in a previous version of the specification. This code is currently not in use. However, the code is reserved for future application.</p>

Message	Description
307 Temporary Redirect	<p>The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client should continue to use the <code>Request-URI</code> for future requests. This response is only cacheable if indicated by a <code>Cache-Control</code> or <code>Expires</code> header field.</p> <p>The temporary URI should be identified by the <code>Location</code> field in the response. Unless the request method was <code>HEAD</code>, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s), since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note should contain the information necessary for a user to repeat the original request on the new URI.</p> <p>If the 307 status code is received in response to a request other than <code>GET</code> or <code>HEAD</code>, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.</p>

D.4 Client Error 4XX

The *4XX* class of status code is intended for cases in which the client seems to have erred. Except when responding to a `HEAD` request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents should display any included entity to the user.

If the client is sending data, a server implementation using TCP should be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

Table D-4: Client Error 4XX message set

Message	Description
400 Bad Request	The request could not be understood by the server due to malformed syntax. The client should not repeat the request without modifications.
401 Unauthorized	The request requires user authentication. The response must include a <code>WWW-Authenticate</code> header field containing a challenge applicable to the requested resource. The client may repeat the request with a suitable <code>Authorization</code> header field. If the request already included <code>Authorization</code> credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user should be presented the entity that was identified in the response, since that entity might include relevant diagnostic information.
402 Payment Required	This code is reserved for future use.
403 Forbidden	The server understood the request, but is refusing to fulfill it. Authorization will not help and the request should not be repeated. If the request method was not <code>HEAD</code> and the server wishes to make public why the request has not been fulfilled, it should describe the reason for the refusal in the entity. If the server does not wish to make this information available to the client, the status code 404 (<code>Not Found</code>) can be used instead.

Message	Description
404 Not Found	The server has not found anything matching the <code>Request-URI</code> . No indication is given of whether the condition is temporary or permanent. The 410 (<code>Gone</code>) status code should be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is essentially a generic, neutral response, commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.
405 Method Not Allowed	The method specified in the <code>Request-Line</code> is not allowed for the resource identified by the <code>Request-URI</code> . The response must include an <code>Allow</code> header containing a list of valid methods for the requested resource.
406 Not Acceptable	<p>The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.</p> <p>Unless it was a <code>HEAD</code> request, the response should include an entity containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type identified in the <code>Content-Type</code> header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. However, this specification does not define any standard for such automatic selection.</p> <p>Note: HTTP/1.1 servers are allowed to return responses which are not acceptable according to the <code>Accept Headers</code> sent in the request. In some cases, this may even be preferable to sending a 406 response. User agents are encouraged to inspect the headers of an incoming response to determine if it is acceptable. If the response could be unacceptable, a user agent should temporarily stop receipt of more data and query the user for a decision on further actions.</p>
407 Proxy Authentication Required	This code is similar to 401 (<code>Unauthorized</code>), but indicates that the client must first authenticate itself with the proxy. The proxy must return a <code>Proxy-Authenticate</code> header field containing a challenge applicable to the proxy for the requested resource. The client may repeat the request with a suitable <code>Proxy-Authorization</code> header field.
408 Request Timeout	The client did not produce a request within the time that the server was prepared to wait. The client may repeat the request without modifications at any later time.

Message	Description
409 Conflict	<p>The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body should include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.</p> <p>Conflicts are most likely to occur in response to a <code>PUT</code> request. For example, if versioning were being used and the entity being <code>PUT</code> included changes to a resource which conflict with those made by an earlier (third-party) request, the server might use the 409 response to indicate that it can't complete the request. In this case, the response entity would likely contain a list of the differences between the two versions in a format defined by the response <code>Content-Type</code>.</p>
410 Gone	<p>The requested resource is no longer available at the server and no forwarding address is known. This condition should be considered permanent. Clients with link editing capabilities should delete references to the <code>Request-URI</code> after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (<code>Not Found</code>) should be used instead. This response is cacheable unless indicated otherwise.</p> <p>The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time—that is left to the discretion of the server owner.</p>
411 Length Required	<p>The server refuses to accept the request without a defined <code>Content-Length</code>. The client may repeat the request if it adds a valid <code>Content-Length</code> header field containing the length of the message-body in the request message.</p>
412 Precondition Failed	<p>The precondition given in one or more of the request-header fields evaluated to <code>False</code> when it was tested on the server. This response code allows the client to place preconditions on the current resource metainformation (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.</p>
413 Request Entity Too Large	<p>The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server may close the connection to prevent the client from continuing the request.</p> <p>If the condition is temporary, the server should include a <code>Retry-After</code> header field to indicate that it is temporary and after what time period has elapsed may the client try again.</p>

Message	Description
414 Request-URI Too Long	The server is refusing to service the request because the <code>Request-URI</code> is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a <code>POST</code> request to a <code>GET</code> request with long query information, when the client has descended into a URI "black hole" of redirection (for example, a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the <code>Request-URI</code> .
415 Unsupported Media Type	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
416 Requested Range Not Satisfiable	<p>A server should return a response with this status code if:</p> <ul style="list-style-type: none"> ▪ A request included a <code>Range</code> request-header field, and ▪ None of the range-specifier values in this field overlap the current extent of the selected resource, and ▪ The request did not include an <code>If-Range</code> request-header field. <p>For byte-ranges, this means that the <code>first-byte-pos</code> of all of the <code>byte-range-spec</code> values were greater than the current length of the selected resource.</p> <p>When this status code is returned for a byte-range request, the response should include a <code>Content-Range</code> entity-header field specifying the current length of the selected resource. This response must not use the <code>multipart/byteranges</code> content-type.</p>
417 Expectation Failed	The expectation identified in an <code>Expect</code> request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

D.5 Server Error 5XX

The 5XX class of status code is intended for cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents should display any included entity to the user. These response codes are applicable to any request method.

Table D-5: Severe Error 5XX message set

Message	Description
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501 Not Implemented	The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.
502 Bad Gateway	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
503 Service Unavailable	<p>The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a <code>Retry-After</code> header. If no <code>Retry-After</code> is given, the client should handle the response as it would for a 500 response.</p> <p>Note: The existence of the 503 status code does not imply that a server must use it when becoming overloaded. Some servers may wish to simply refuse the connection.</p>
504 Gateway Timeout	<p>The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP, LDAP) or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.</p> <p>Note: Implementers should note that some deployed proxies are known to return 400 or 500 when DNS lookups time out.</p>
505 HTTP Version Not Supported	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client. The response should contain an entity describing why that version is not supported and what other protocols are supported by that server.

7

7Bit flag · R-425, R-429, R-444, R-447

8

8Bit flag · R-425, R-429, R-444, R-447
 8BitBinary flag · R-425, R-429, R-444, R-447

A

action - form property · R-19
 ActiveX object · P-194
 ActiveXObject() constructor · P-196
 (for RDS) · P-203
 automatic conversion to JavaScript data
 types · P-199
 casting functions · P-201
 interfaces

 IDispatch · P-195
 TypeInfo · P-195
 IUnknown · P-195
 Remote Data Service (RDS) support · P-203
 example · P-204
 ActiveX objects · P-103
 ActiveXObject object
 ActiveXObject() constructor · P-196
 ActiveXObject() constructor · P-196
 (for RDS) · P-203
 Add()
 wlGeneratorGlobal method · R-21
 wlSystemGlobal method · R-21
 AddAttachment()
 wlNNTP method · R-429
 wlSMTP method · R-447
 adding
 comments to an Agenda · P-23
 JavaScript object nodes, to Agenda Tree · P-
 75
 transactions · P-50
 addProperty() - WSCComplexObject method · R-
 22

- AdjacentText - element property · R-23
- Agenda Tree · P-64
 - navigation blocks · R-13
 - nodes · P-64
 - structure · P-64
- Agendas
 - creating · P-6, R-4, P-63
 - examples
 - basic, recorded with AAT (protocol mode) · P-240
 - cookies, setting · P-242
 - input/output files used in an Agenda (protocol mode) · P-241
 - parsing dynamic HTML (protocol mode) · P-242
 - parsing dynamic links (protocol mode) · P-242
 - random number generator · P-242
 - testing a customer service site · P-11
 - timers · P-242
 - execution sequence
 - after a runtime error · P-111
 - mixed clients · P-70
 - normal · P-67
 - scheduled clients · P-69
 - initialization and termination functions · P-66
 - main script · P-66, P-68
 - navigation block · R-13, P-18, P-65
 - synchronization points · P-48
 - threads · P-68
 - what are Agendas · P-6, R-4, P-63
- Alert - wIBrowser property (dialog box) · R-75
- Alt
 - area property · R-24
 - element property · R-24
 - image property · R-24
- Append() - wIFTP method · R-402
- AppendFile() - wIFTP method · R-403
- area object · R-24
 - properties
 - Alt · R-24
 - ContainingMap · R-56
 - Coords · R-57
 - id · R-144
 - Shape · R-246
 - URL · R-303
- ArticleText - wINNTTP property · R-425
- ASCII files, reading input with GetLine() function · R-125, P-86
- ASYNCH_PLAY flag · R-201
- Attachments
 - wINNTTP property · R-425
 - wSMTP property · R-444
- AttachmentsEncoding
 - wINNTTP property · R-425
 - wSMTP property · R-444
- AttachmentsTypes
 - wINNTTP property · R-426
 - wSMTP property · R-445
- authentication
 - certificates (protocol mode) · R-260
 - of users, with NT Challenge Response protocol (protocol mode) · R-187
 - protocol options · R-66
- AutoDelete - wIPOP property · R-436
- Automatic State Management (ASM)
 - dynamic session IDs (protocol mode) · P-225
 - example
 - dynamic session ID in FormData (protocol mode) · P-244
 - identifying dynamic session IDs (protocol mode) · P-228
 - working with session ID (protocol mode) · P-246
 - functions
 - GetCurrentValue() - wHttp method (protocol mode) · P-227
 - IdentifyObject() - wHttp method (protocol mode) · P-227
 - setting dynamic values (protocol mode) · P-227
 - target frame (protocol mode) · P-225

visual set (protocol mode) · P-225
 wHttpRequest properties
 complete list (protocol mode) · P-225
 ExpectedDocument (protocol mode) · R-93
 ExpectedDOMID (protocol mode) · R-94
 ExpectedID (protocol mode) · R-95
 ExpectedLocation (protocol mode) · R-95
 ExpectedName (protocol mode) · R-96
 ExpectedText (protocol mode) · R-97
 wITarget (protocol mode) · R-366
 wHttpRequest properties - complete list (protocol mode) · R-25
 AutoNavigate() - wIBrowser method · R-26

B

Back() - wIBrowser method · R-27
 Base64 flag · R-425, R-429, R-444, R-447
 Bcc - wISMTP property · R-445
 BeginTransaction() function · R-28, P-141
 example · P-50
 Bind() - wIUDP method · R-466
 bitrate - wIMediaPlayer property · R-30
 Broadcast() - wIUDP method · R-466
 Browser configuration
 executing commands · P-115
 transaction-specific · P-115
 wIGlobals, wIHttpRequest, wILocals property set · R-31
 BrowserCache - wIGlobals, wILocals, wIHttpRequest property (protocol mode) · R-33
 Button
 document property · R-34
 object · R-34
 properties
 InnerImage · R-154
 InnerText · R-156

OnClick · R-190
 title · R-296

C

cache, SSL, cleared at round's end · P-69
 CBool() function · P-201
 CByte() function · P-201
 Cc - wISMTP property · R-445
 CDbf() function · P-201
 cell object
 (protocol mode) · R-35
 properties
 cellIndex · R-37
 InnerHTML · R-153
 InnerText · R-156
 tagName · R-289
 cellIndex - cell property · R-37
 cells
 row property · R-35
 wITable property · R-35
 certificates, submitting (protocol mode) · R-260
 CFlt() function · P-201
 ChangeDir() - wIFTP method · R-403
 Checkbox
 document property · R-38
 object · R-38
 checked - element property · R-40
 ChFileMod() - wIFTP method · R-403
 Child Transaction Instance Tree (fifth level) · P-156
 ChMod() - wIFTP method · R-404
 CInt() function · P-201
 Cipher Command Suite · R-256
 ClearAll() - wICookie method · R-40
 ClearDNSCache() - wIHttpRequest method (protocol mode) · R-41
 ClearSSLCache() - wIHttpRequest method (protocol mode) · R-42
 Client Types (Thick, Thin, Normal) · P-104

- ClientAuthentication - wlBrowser property (dialog box) · R-75
- ClientCertificate - wlBrowser property (dialog box) · R-75
- ClientNum variable · P-57, R-43
- clients
 - example, identifying clients · P-58, R-146
 - mixed · P-70
 - Probing and Virtual · P-9
 - scheduled · P-69
 - Types (Thick, Thin, Normal) · P-104
 - unique number · P-57, R-43
 - virtual, VCUniqueID() identification function · R-312
- CLng() function · P-201
- Close()
 - function · R-45
 - wlOutputFile method · R-45
- CloseConnection() - wlHttp method (protocol mode) · R-46
- closing external files · R-45
- collections · R-47
 - elements · R-83
 - forms · R-103
 - frames · R-107
 - images · R-148
 - length property · R-167
 - links · R-169
 - methods
 - wlGet() (data - protocol mode) · R-335
 - wlSet() (data - protocol mode) · R-359
 - options · R-195
 - scripts · R-229
 - wlHeaders · R-341
 - wlMetas · R-346
 - wlSearchPairs · R-357
 - wlTables · R-364
 - wlXmIs · R-371
- cols
 - element property · R-48
 - wlTable property · R-48
- comments - adding to an Agenda · P-23
- Compare() - TableCompare method · R-49
- CompareColumns - TableCompare property · R-51
- CompareRows - TableCompare property · R-52
- complex types · P-207
- Component Object Model (COM) · P-194
 - ActiveXObject() constructor · P-196 (for RDS) · P-203
 - automatic conversion to JavaScript data types · P-199
 - casting functions · P-201
 - interfaces
 - IDispatch · P-195
 - ITypeInfo · P-195
 - IUnknown · P-195
 - Remote Data Service (RDS) support · P-203 example · P-204
- configuration properties
 - search order precedence · P-127
 - setting with wlHttp, wlLocals, and wlGlobals objects, example (protocol mode) · P-253
- Confirm - wlBrowser property (dialog box) · R-75
- Connect()
 - wlIMAP method · R-415
 - wlNNTP method · R-429
 - wlPOP method · R-438
 - wlSMTP method · R-447
 - wlTCP method · R-453
 - wlTelnet method · R-459
- connectionBandwidth - wlMediaPlayer property · R-53
- connections, closing · R-46, P-69
- ConnectionSpeed - wlGlobals property (protocol mode) · R-54
- constructors
 - ActiveXObject() · P-196
 - ActiveXObject() (for RDS) · P-203
 - TableCompare() · R-287

- wlException() · R-333
 - WLFtp() · R-409
 - WLImap() · R-420
 - wlMediaPlayer() · R-345
 - WLNntp() · R-433
 - wlOutputFile() · R-354
 - WLPop() · R-441
 - WLSmtp() · R-449
 - WLTcp() · R-455
 - WLTelnet() · R-460
 - WLUdp() · R-468
 - WLXmlDocument() - wLXmIs object · R-369
 - WSComplexObject() · R-375
 - WSWebService() · R-379
 - ContainerTag - UIContainer property · R-55
 - ContainingMap - area property · R-56
 - content - wlMeta property · R-56
 - context
 - global · P-118
 - limited · P-115
 - local · P-116
 - Cookie - wlBrowser property (dialog box) · R-75
 - cookies
 - deleted at round's end · P-69
 - example · R-331, P-239
 - example, setting a cookie · P-242
 - how WebLOAD works with · R-330, P-238
 - Coords - area property · R-57
 - CopyFile() function · R-57, P-81
 - copying files · R-57
 - search order precedence · P-82
 - counters, automatic, for Java class methods · P-187
 - CreateDOM() function · R-59, P-150
 - CreateMailbox() - wIIMAP method · R-416
 - CreateTable() function · R-61
 - creating
 - a collection of expected tables · R-61
 - an Agenda · P-6, R-4, P-63
 - an expected DOM · R-59, P-150
 - cryptology
 - cryptographic strength, defining (protocol mode) · R-262
 - function list · R-256
 - SSLBitLimit - wlGlobals property (protocol mode) · R-257
 - SSLCipherSuiteCommand() function · R-258
 - SSLClientCertificateFile - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-260
 - SSLClientCertificatePassword - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-260
 - SSLCryptoStrength - wlGlobals property (protocol mode) · R-262
 - SSLDisableCipherID() function · R-263
 - SSLDisableCipherName() function · R-265
 - SSLEnableCipherID() function · R-266
 - SSLEnableCipherName() function · R-267
 - SSLGetCipherCount() function · R-268
 - SSLGetCipherID() function · R-269
 - SSLGetCipherInfo() function · R-270
 - SSLGetCipherName() function · R-271
 - SSLGetCipherStrength() function · R-272
 - SSLUseCache - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-274
 - SSLVersion - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-275
 - CurrentMessage - wIIMAP property · R-413
 - CurrentMessageID - wIIMAP property · R-413
 - currentPosition - wlMediaPlayer property · R-62
 - currentStreamName - wlMediaPlayer property · R-63
 - customer service site, example · P-11
 - CVARIANT() function · P-201
- ## D
- data
 - files, reading · R-125
 - global, user-defined properties · R-305

- HTTP submission properties (protocol mode)
 - P-229
- input, from external files · R-125, P-86
- Data
 - wlFTP property · R-400
 - wlHttp property (protocol mode) · R-63, P-232
- Data Drilling
 - Child Transaction Instance Tree (fifth level) · P-156
 - Parent Transaction Instance Tree (fourth level) · P-155
 - reports · P-151
 - TableCompare Results (WebLOAD Viewer) · P-160
 - TableCompare Results Tree (three levels) · P-157
 - Transaction Failure Reason Grid (second level) · P-153
 - Transaction Grid (first level) · P-152
 - Transaction Instance Grid (third level) · P-154
- Data Islands · P-163
- DataFile
 - wlFTP property · R-400
 - wlHttp property (protocol mode) · R-65, P-233
- debugging transaction failures · P-151
- DefaultAuthentication - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-66
- defaultchecked - element property · R-68
- defaults, for global and local browser actions · P-127
- defaultselected - option property · R-69
- defaultvalue - element property · R-69
- defining
 - cryptographic strength (protocol mode) · R-262
 - transactions · P-50
- Delete()
 - wlCookie method · R-70
 - wlFTP method · R-404
 - wlIMAP method · R-416
 - wlPOP method · R-439
- delete() - wlOutputFile method · R-70
- DeleteAttachment()
 - wlNNTP method · R-430
 - wlSMTP method · R-448
- DeleteFile() - wlFTP method · R-404
- DeleteMailbox() - wlIMAP method · R-416
- Details - TableCompare property · R-72
- dialog boxes · R-73, P-77
 - wlBrowser properties, complete list · R-73
- Dir() - wlFTP method · R-405
- DisableAll flag · R-258
- DisableSleep - wlBrowser property · R-75
- Disconnect()
 - wlIMAP method · R-416
 - wlNNTP method · R-430
 - wlPOP method · R-439
 - wlSMTP method · R-448
 - wlTCP method · R-454
 - wlTelnet method · R-459
- displaying
 - error messages · R-89
 - information messages · R-152
 - severe error messages · R-245
 - warning messages · R-323
- Div
 - document property · R-77
 - object · R-77
 - properties
 - InnerText · R-156
 - OnClick · R-190
 - OnMouseOver · R-192
 - title · R-296
- DNSUseCache - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-78
- document
 - object · R-79
 - example (protocol mode) · P-222
 - methods

- wlGetAllForms() (protocol mode) · R-337
- wlGetAllFrames() (protocol mode) · R-337
- wlGetAllLinks() (protocol mode) · R-338
- properties
 - Button · R-34
 - Checkbox · R-38
 - Div · R-77
 - File · R-99
 - forms · R-103
 - frames · R-107
 - images · R-148
 - InputButton · R-157
 - InputCheckbox · R-158
 - InputFile · R-159
 - InputImage · R-160
 - InputPassword · R-161
 - InputRadio · R-163
 - InputText · R-164
 - links · R-169
 - location · R-176
 - Radiobutton · R-210
 - Reset · R-218
 - scripts · R-229
 - Select · R-232
 - Span · R-254
 - Submit · R-279
 - Text · R-291
 - TextArea · R-292
 - title · R-296
 - wlHeaders (protocol mode) · R-340
 - wlMetas (protocol mode) · R-346
 - wlSource (protocol mode) · R-361
 - wlStatusLine (protocol mode) · R-361
 - wlStatusNumber (protocol mode) · R-362
 - wlTables (protocol mode) · R-364
 - wlVersion (protocol mode) · R-368
 - wlXmIs · R-370
 - window property · R-79
 - wlFTP property · R-400
 - wlIMAP property · R-413
 - wlNNTP property · R-426
 - wlPOP property · R-436
 - wlTCP property · R-451
 - wlTelnet property · R-457
 - wlUDP property · R-463
- Document Object Model (DOM)
 - basic structure · R-9
 - creating an expected DOM · P-150
 - document object · R-9
 - element object · R-10
 - form object · R-10
 - frame object · R-9
 - image object · R-10
 - input object · R-10
 - link object · R-10
 - location object · R-10
 - script object · R-10
 - title object · R-10
 - WebLOAD extensions
 - complete set · R-10, P-217
 - structure · P-217
 - window object · R-9
- DOM, expected · R-59
- DOR - Dynamic Object Recognition · P-97
- Download() - wlFTP method · R-405
- DownloadFile() - wlFTP method · R-405
- DTD, in XML document, example · P-178
- duration - wlMediaPlayer property · R-80
- dynamic HTML
 - Automatic State Management (ASM) (protocol mode) · P-225
 - dynamic session ID in FormData, example using Automatic State Management (ASM) (protocol mode) · P-244
 - example, parsing (protocol mode) · P-242
 - example, parsing and navigating nested frames (protocol mode) · P-222

parsing · R-4
 dynamic link example (protocol mode) · P-223
 Dynamic Object Recognition (DOR) · P-97
 wIbrowsers properties - complete list · R-81
 dynamic session IDs · R-4

E

element object · R-83
 example (protocol mode) · P-236
 properties
 AdjacentText · R-23
 Alt · R-24
 checked · R-40
 cols · R-48
 defaultchecked · R-68
 defaultvalue · R-69
 id · R-144
 InnerImage · R-154
 InnerText · R-156
 MaxLength · R-180
 Name · R-184
 options · R-195
 rows · R-223
 selectedIndex · R-234
 Size · R-247
 title · R-296
 type · R-300
 URL · R-303
 value · R-310
 elements
 collection · R-83
 form property · R-83
 EnableAll flag · R-258
 ENC_7BIT flag · R-426, R-444
 ENC_8BIT flag · R-426, R-444
 ENC_8BITBINARY flag · R-426, R-444
 ENC_BASE64 flag · R-426, R-444
 ENC_QUOTED flag · R-426, R-444
 encoding - form property · R-85
 EndTransaction() function · R-85, P-141
 example · P-51
 Erase - wIHttp property (protocol mode) · R-87,
 P-116, P-234
 Erase()
 wITCP method · R-454
 wITelnet method · R-459
 wIUDP method · R-466
 error
 handling · P-110, P-111, P-214
 messages, displaying · P-24, R-180
 ErrorMessage() function · R-89
 EvaluateScript() function · R-90
 event
 OnClick · R-190
 OnMouseOver · R-192
 script property · R-92
 TableCell property · R-92
 UIContainer property · R-92
 Examples
 Automatic State Management (ASM)
 dynamic session ID in FormData
 (protocol mode) · P-244
 identifying dynamic session IDs
 (protocol mode) · P-228
 working with session ID (protocol mode)
 · P-246
 automatic timers and counters · P-187
 building a complete database in XML · P-
 176
 calling a WebLOAD API · P-189
 connecting to a secure internet server
 (protocol mode) · P-254
 connecting to server using SSL (protocol
 mode) · P-253
 cookies, setting · P-242
 Data Islands used in an Agenda · P-167
 DTD in XML document · P-178
 dynamic HTML page, parsing, and
 navigating nested frames (protocol
 mode) · P-222

- dynamic HTML parsing (protocol mode) · P-242
 - generating a random number · R-355
 - identifying clients and rounds · P-58, R-146
 - input/output files used in an Agenda (protocol mode) · P-241
 - parse tree illustration (protocol mode) · P-221
 - parsing links in a dynamic HTML page (protocol mode) · P-242
 - passing objects between Java and JavaScript · P-186
 - passing simple variables between Java and JavaScript · P-185
 - posting form data using elements (protocol mode) · P-236
 - random number generator · P-242
 - reading data from a JDBC database · P-191
 - reading input files · P-89
 - remote ActiveX object access · P-204
 - searching with key-value pairs (HTTP header parsing example) (protocol mode) · R-341, P-255
 - searching with key-value pairs (link parsing) · R-357
 - setting configuration properties in wLHttp, wLocals, and wGlobals objects (protocol mode) · P-253
 - testing a customer service site · P-11
 - timers · P-242
 - transaction verification using TableCompare() constructor (protocol mode) · P-249
 - transactions · P-50
 - travel agency (recorded Agenda) (protocol mode) · P-240
 - using a cookie · R-331, P-239
 - using a timer to report statistics · R-295
 - using HTML properties of the XML DOM in an Agenda · R-371
 - using synchronization points · R-282
 - using wLHtml to follow a dynamic link (protocol mode) · P-223
 - web page illustration (protocol mode) · P-220
 - working with location objects (protocol mode) · P-251
 - working with messages · R-180
 - working with option objects (protocol mode) · P-252
 - ExceptionEnabled - wLBrowser property · R-92
 - expected DOM · R-59, P-150
 - expected tables, collection · R-61
 - ExpectedDocument - wLHttp property (protocol mode) · R-93
 - ExpectedDOMID - wLHttp property (protocol mode) · R-94
 - ExpectedID - wLHttp property (protocol mode) · R-95
 - ExpectedLocation - wLHttp property (protocol mode) · R-95
 - ExpectedName - wLHttp property (protocol mode) · R-96
 - ExpectedText - wLHttp property (protocol mode) · R-97
 - ExpectNavigation()
 - with named transactions · R-97
 - wLBrowser method · R-97
- ## F
- File
 - document property · R-99
 - object · R-99
 - file management functions · R-100
 - Close() · R-45
 - CopyFile() · R-57, P-81
 - GetLine() · R-125, P-86
 - IncludeFile() · R-150, P-79
 - Open() · R-124, R-193, P-87
 - See wLOutputFile object · R-100
 - FileName - wLHttp.DataFile property · R-101
 - fileName - wLMediaPlayer property · R-101
 - files

- closing external · R-45
- copying · P-81
- functions
 - Close() · R-45
 - CopyFile() · P-81
 - EvaluateScript() · R-90
 - GetLine() · R-125, P-86
 - IncludeFile() · R-150, P-79
 - Open() · R-193, P-87
 - See wIOutputFile object · R-351
- including · R-90, R-150, P-79
- input · P-36, P-85
- opening external · R-193
- output · R-351, P-83
- reading ASCII data · R-125, P-86
- FindObject() - wIBrowser method · R-102
- flags
 - 7Bit · R-425, R-429, R-444, R-447
 - 8Bit · R-425, R-429, R-444, R-447
 - 8BitBinary · R-425, R-429, R-444, R-447
 - ASYNCH_PLAY · R-201
 - Base64 · R-425, R-429, R-444, R-447
 - DisableAll · R-258
 - EnableAll · R-258
 - ENC_7BIT · R-426, R-444
 - ENC_8BIT · R-426, R-444
 - ENC_8BITBINARY · R-426, R-444
 - ENC_BASE64 · R-426, R-444
 - ENC_QUOTED · R-426, R-444
 - INFINITE_PLAY · R-201
 - Quoted · R-425, R-429, R-444, R-447
 - ShowAll · R-258
 - ShowEnabled · R-258
 - SSL_AllCrypto · R-262
 - SSL_ExportCryptoOnly · R-262
 - SSL_ServerGatedCrypto · R-262
 - SSL_StrongCryptoOnly · R-262
 - WLAfterInitAgenda · R-90
 - WLAfterRound · R-90
 - WLAfterTerminateAgenda · R-90
 - WLAfterTerminateClient · R-90
 - WLAllAgendas · R-21, R-110, R-237, P-123, P-125
 - WLBeforeInitClient · R-90
 - WLBeforeRound · R-90
 - WLBeforeThreadActivation · R-90
 - WLCurrentAgenda · R-21, R-110, R-237, P-123, P-125
 - WLError · P-49, P-52, R-86, R-133, R-281, R-294, R-309, R-320, R-333, P-108
 - WLExecuteScript · R-150
 - WLLoadChanged · P-49, R-281
 - WLMinorError · P-52, R-86, R-294, R-309, R-320, P-108
 - WLOnThreadActivation · R-90
 - WLRandom · R-124, R-193, P-86
 - WLSequential · R-124, R-193, P-86
 - WLSevereError · P-52, R-86, R-133, R-294, R-309, R-320, R-333, P-108
 - WLSuccess · P-49, P-52, R-86, R-281, R-294, R-309, R-320, P-108
 - WTimeout · P-49, R-281
- flash objects · P-103
- Form Data Wizard · P-36
- form object · R-103
 - methods
 - FindObject() · R-102
 - wISubmit() · R-362
 - properties
 - action · R-19
 - elements · R-83
 - encoding · R-85
 - id · R-144
 - method · R-182
 - Name · R-184
 - target · R-290
 - URL · R-303
- FormData - wIHttp property
 - missing fields (protocol mode) · P-230
 - using Get() (protocol mode) · R-105, P-229
 - using Post() (protocol mode) · R-105, P-230

- working with data files (protocol mode) · P-231
- forms
 - collection · R-103
 - document property · R-103
- Forward() - wIBrowser method · R-106
- frames
 - collection · R-107
 - properties
 - length · R-167
 - window · R-324
 - document property · R-107
 - object
 - properties
 - id · R-144
 - index · R-151
 - Name · R-184
 - title · R-296
 - URL · R-303
 - window property · R-107
- From
 - wINNTTP property · R-427
 - wISMTTP property · R-445
- FTP
 - File Transfer Protocol · R-400
 - Sample Code · R-410
- FtpAuthentication - wIBrowser property (dialog box) · R-75
- functional testing
 - BeginTransaction() function · R-28, P-141
 - complete function list · R-297
 - CreateDOM() function · R-59, P-150
 - CreateTable() function · R-61
 - creating an expected DOM · P-150
 - Data Drilling
 - Child Transaction Instance Tree (fifth level) · P-156
 - Parent Transaction Instance Tree (fourth level) · P-155
 - reports · P-151
 - TableCompare Results (WebLOAD Viewer) · P-160
 - TableCompare Results Tree (three levels) · P-157
 - Transaction Failure Reason Grid (second level) · P-153
 - Transaction Grid (first level) · P-152
 - Transaction Instance Grid (third level) · P-154
 - EndTransaction() function · R-85, P-141
 - example using TableCompare() constructor (protocol mode) · P-249
 - named HTTP transactions (protocol mode) · R-112, R-203, P-139
 - named navigation activities · R-97
 - ReportEvent() function · R-214
 - SetFailureReason() function · R-240, P-142
 - TableCompare object · P-145
 - user-defined transaction name · R-28, R-85
 - VerificationFunction() (user-defined function) · R-320, P-141
- functions
 - ASM
 - GetCurrentValue() - wIHttp method (protocol mode) · P-227
 - IdentifyObject() - wIHttp method (protocol mode) · P-227
 - COM interface
 - CBool() · P-201
 - CByte() · P-201
 - CDbl() · P-201
 - Cflt() · P-201
 - CInt() · P-201
 - CLng() · P-201
 - CVARIANT() · P-201
 - file management · R-100
 - Close() · R-45
 - CopyFile() · R-57, P-81
 - GetLine() · R-125, P-86
 - IncludeFile() · R-150, P-79
 - Open() · R-124, R-193, P-87
 - See wIOutputFile object · R-351

- identification · P-56, R-146
 - GeneratorName() · R-108
 - GetOperatingSystem() · R-130
 - VCUniqueID() · R-312
 - initialization and termination · P-66
 - EvaluateScript() · R-90
 - InitAgenda() · P-68
 - InitClient() · P-68
 - OnErrorTerminateAgenda() · P-111
 - OnErrorTerminateClient() · P-111
 - OnScriptAbort() · P-110, P-111
 - TerminateAgenda() · P-68
 - TerminateClient() · P-68
 - messages · P-24, R-180
 - ErrorMessage() · R-89
 - InfoMessage() · R-152
 - SevereErrorMessage() · R-245
 - WarningMessage() · R-323
 - security
 - complete list · R-256
 - SSLCipherSuiteCommand() · R-258
 - SSLDisableCipherID() · R-263
 - SSLDisableCipherName() · R-265
 - SSLEnableCipherID() · R-266
 - SSLEnableCipherName() · R-267
 - SSLGetCipherCount() · R-268
 - SSLGetCipherID() · R-269
 - SSLGetCipherInfo() · R-270
 - SSLGetCipherName() · R-271
 - SSLGetCipherStrength() · R-272
 - sleeping · P-29
 - SynchronizationPoint() · P-48, R-281
 - timers · P-27, R-295
 - SendCounter() · R-234
 - SendMeasurement() · R-235
 - SendTimer() · R-236
 - SetTimer() · R-241
 - Sleep() · R-247
 - transaction verification
 - BeginTransaction() · R-28, P-141
 - complete list · R-297
 - CreateDOM() · R-59, P-150
 - CreateTable() · R-61
 - EndTransaction() · R-85, P-141
 - ReportEvent() · R-214
 - SetFailureReason() · R-240, P-142
 - VerificationFunction() (user-defined) · R-320, P-141
 - WebServices
 - WSGetSimpleValue() · R-376
- G**
- GeneratorName() function · R-108
 - Get()
 - with named transactions (protocol mode) · R-112
 - wlGeneratorGlobal method · R-110
 - wlHttp method (protocol mode) · R-111
 - wlSystemGlobal method · R-110
 - GetArticle() - wlNNTP method · R-430
 - GetArticleCount() - wlNNTP method · R-431
 - GetCurrentMessageID() - wlPOP method · R-439
 - GetCurrentPath() - wlFTP method · R-406
 - GetCurrentValue() - wlHttp method (protocol mode) · R-114, P-227
 - GetFieldValue() - wlHtml method (protocol mode) · R-115
 - GetFieldValueInForm() - wlHtml method (protocol mode) · R-116
 - GetFormAction() - wlHtml method (protocol mode) · R-117
 - GetFrameByUrl() - wlHtml method (protocol mode) · R-118
 - GetFrameUrl() - wlHtml method (protocol mode) · R-119
 - GetHeaderValue() - wlHtml method (protocol mode) · R-120
 - GetHost() - wlHtml method (protocol mode) · R-121

GetHostName() - wHtml method (protocol mode) · R-122
 GetIPAddress() - wHttp method (protocol mode) · R-123
 GetLine() function · R-125, P-86
 GetLinkByName() - wHtml method (protocol mode) · R-127
 GetLinkByUrl() - wHtml method (protocol mode) · R-128
 GetMailboxSize() - wPOP method · R-439
 GetMessage() - wException method · R-129
 GetMessageCount()
 wIMAP method · R-417
 wPOP method · R-440
 GetOperatingSystem() function · R-130
 GetPortNum() - wHtml method (protocol mode) · R-131
 GetQSFieldValue() - wHtml method (protocol mode) · R-132
 GetSeverity() - wException method · R-133
 GetStatusLine()
 wFTP method · R-406
 wHtml method (protocol mode) · R-134
 wIMAP method · R-417
 wNNTP method · R-431
 wPOP method · R-440
 GetStatusNumber() - wHtml method (protocol mode) · R-135
 GetUri() - wHtml method (protocol mode) · R-136
 global
 browser configuration defaults · P-127
 context · P-118
 data, user defined · R-305
 objects · P-118, P-119
 variables, user-defined · P-118, P-121, P-123, P-125
 Group - wNNTP property · R-427
 GroupOverview() - wNNTP method · R-431

H

hash - link and location property · R-137
 Head() - wHttp method (protocol mode) · R-138
 Header
 object
 properties
 key · R-166
 wHttp property (protocol mode) · R-138, P-233
 Headers[] - wPOP property · R-437
 HistoryLimit - wGlobals property · R-140
 host - link and location property · R-141
 hostname - link and location property · R-141
 href - link and location property · R-142
 HTTP
 Automatic State Management (ASM)
 properties (protocol mode) · P-225
 configuration
 properties, setting & erasing (protocol mode) · P-116
 data submission properties (protocol mode) · P-229
 error handling · P-110
 header (protocol mode) · R-341, P-255
 httpEquiv - wMeta property · R-143
 HTTPS protocol (protocol mode) · R-275

I

id
 area property · R-144
 element property · R-144
 form property · R-144
 frames property · R-144
 image property · R-144
 link property · R-144
 location property · R-144
 map property · R-144

- script property · R-144
- table property · R-144
- TableCell property · R-144
- UIContainer property · R-144
- wlTable property · R-144
- wlXmIs property · R-144
- identification
 - functions · P-56, R-146
 - GeneratorName() · R-108
 - GetOperatingSystem() · R-130
 - VCUniqueID() · R-312
 - individual clients · P-57, R-43
 - number of current round · P-57, R-221
 - variables · P-56, R-146
 - ClientNum · P-57, R-43
 - RoundNum · P-57, R-221
- IdentifyObject() - wlHttp method (protocol mode) · R-147, P-227
- IDispatch (COM interface) · P-195
- image object · R-148
 - properties
 - Alt · R-24
 - id · R-144
 - InnerLink · R-155
 - Name · R-184
 - OuterLink · R-197
 - protocol · R-208
 - src · R-255
 - title · R-296
 - URL · R-303
- images
 - collection · R-148
 - document property · R-148
- IMAP
 - Internet Message Access Protocol · R-413
 - Sample Code · R-421
- InBufferSize
 - wlTCP property · R-451
 - wlUDP property · R-463
- IncludeFile() function · R-150, P-79
- including
 - files, search order precedence · P-80
 - functions, from external files · R-90, R-150
- index - frames property · R-151
- INFINITE_PLAY flag · R-201
- InfoMessage() function · R-152
- InitAgenda() function · P-68
- InitClient() function · P-68
- initialization functions · P-66
 - EvaluateScript() function · R-90
 - InitAgenda() function · P-68
 - InitClient() function · P-68
- initializing an Agenda · P-68
- innerHTML
 - script property · R-153
 - wlXmIs property · R-153
- InnerHTML - cell property · R-153
- InnerImage
 - button property · R-154
 - element property · R-154
 - link property · R-154
 - location property · R-154
 - TableCell property · R-154
 - UIContainer property · R-154
- InnerLink - image property · R-155
- InnerText
 - Button property · R-156
 - cell property · R-156
 - Div property · R-156
 - element property · R-156
 - link property · R-156
 - location property · R-156
 - Span property · R-156
 - TableCell property · R-156
 - UIContainer property · R-156
- input files · P-36, P-85
 - example · P-89
 - functions
 - GetLine() · P-86
 - Open() · P-87

I/O files used in an Agenda, example
 (protocol mode) · P-241
 reading an ASCII file · P-86
 requirements · P-86

input object
 methods
 wIClick() · R-326
 wIMouseDown() · R-347
 wIMouseOver() · R-348
 wIMouseUp() · R-349
 wIMultiSelect() · R-350
 wISelect() · R-358
 wITypeIn() · R-367

InputButton
 document property · R-157
 object · R-157
 properties
 title · R-296
 value · R-310

InputCheckbox
 document property · R-158
 object · R-158
 properties
 title · R-296
 value · R-310

InputFile
 document property · R-159
 object · R-159

InputImage
 document property · R-160
 object · R-160

InputPassword
 document property · R-161
 object · R-161

InputRadio
 document property · R-163
 object · R-163
 properties
 value · R-310

InputText

document property · R-164
 object · R-164

IP · See MultiIPSupport, wIGlobals property

IPP · See WebLOAD Internet Productivity Pack

ITypeInfo (COM interface) · P-195

IUnknown (COM interface) · P-195

J

Java
 accessing Java objects from JavaScript
 Agendas · P-183
 automatic timers and counters, example · P-187
 calling a WebLOAD API, example · P-189
 JavaScript interface · P-180
 LiveConnect interface · P-257
 passing objects between Java and JavaScript,
 example · P-186
 passing simple variables between Java and
 JavaScript, example · P-185
 reading data from a JDBC database, example
 · P-191
 requirements · P-181

Java Applet objects · P-103

JavaScript
 adding object nodes to the Agenda Tree · P-75
 automatic conversion to COM (ActiveX)
 data types · P-199
 language · R-4, P-64
 programming
 advanced features · P-161

K

KeepAlive - wIGlobals, wILocals, wIHttp
 property (protocol mode) · R-165

key

Header property · R-166
 wlHeader property · R-166
 wlSearchPair property · R-166

L

language - script property · R-167
 length - collection property · R-167
 limited
 context · P-115
 objects · P-115
 variables, user-defined · P-115
 LineArray object · R-125, P-86
 link object · R-169
 properties
 hash · R-137
 host · R-141
 hostname · R-141
 href · R-142
 id · R-144
 InnerImage · R-154
 InnerText · R-156
 Name · R-184
 OnClick · R-190
 OnMouseOver · R-192
 pathname · R-199
 port · R-202
 protocol · R-208
 search · R-230
 target · R-290
 title · R-296
 URL · R-303
 wlSearchPairs · R-357
 links
 collection · R-169
 document property · R-169
 example
 parsing dynamic links (protocol mode) · P-242

 parsing key-value pairs · R-357
 ListGroups() - wInNTP method · R-431
 ListLocalFiles() - wIFTP method · R-406
 ListMailboxes() - wIMAP method · R-417
 Load Generators · P-8
 GeneratorName() function · R-108
 GetOperatingSystem() function · R-130
 load()
 wIXmls method · R-171
 XML DOM method · P-175
 LoadGeneratorThreads - wIGlobals, wILocals, wIHttp property (protocol mode) · R-173
 loadXML()
 wIXmls method · R-175
 XML DOM method · P-175
 local
 browser configuration defaults · P-127
 context · P-116
 HTTP transactions
 configuration defaults (protocol mode) · R-343
 objects · P-116
 variables, user-defined · P-116
 LocalHost - wIUDP property · R-463
 LocalPort
 wITCP property · R-451
 wIUDP property · R-464
 location
 document property · R-176
 object
 example (protocol mode) · P-251
 properties · See link object property list
 hash · R-137
 host · R-141
 hostname · R-141
 href · R-142
 id · R-144
 InnerImage · R-154
 InnerText · R-156
 Name · R-184
 pathname · R-199

- port · R-202
- protocol · R-208
- search · R-230
- selectedIndex · R-234
- title · R-296
- URL · R-303
- wlSearchPairs · R-357
- window property · R-176
- logical transactions · P-50
- Logoff() - wLFTP method · R-407
- Logon() - wLFTP method · R-407

M

- Mailbox - wLIMAP property · R-414
- main script
 - in Agendas · P-66
 - running repeatedly · P-68
- MakeDir() - wLFTP method · R-407
- map object · R-178
 - properties
 - id · R-144
 - Name · R-184
- MatchBy - TableCompare property · R-178
- matching text key/value pairs
 - with wlHeader (protocol mode) · R-340
 - with wlSearchPair · R-356
- MaxDatagramSize - wLUDP property · R-464
- MaxHeadersLength - wLNNTP property · R-427
- MaxLength - element property · R-180
- MaxLines
 - wLIMAP property · R-414
 - wLPOP property · R-437
- measuring performance times · P-27, R-295
- Message - wLSMTP property · R-446
- messages
 - displaying
 - errors · R-89
 - information · R-152
 - on Console · P-24, R-180
 - severe errors · R-245
 - warning messages · R-323
- ErrorMessage() function · R-89
- example · R-180
- InfoMessage() function · R-152
- SevereErrorMessage() function · R-245
- WarningMessage() function · R-323
- method - form property · R-182
- methods
 - ActiveXObject() constructor · P-196
 - (for RDS) · P-203
 - Add()
 - wLGeneratorGlobal method · R-21
 - wLSystemGlobal method · R-21
 - AddAttachment()
 - wLNNTP method · R-429
 - wLSMTP method · R-447
 - addProperty() - WSCOMPLEXOBJECT method · R-22
 - Append() - wLFTP method · R-402
 - AppendFile() - wLFTP method · R-403
 - AutoNavigate() - wLBROWSER method · R-26
 - Back() - wLBROWSER method · R-27
 - Bind() - wLUDP method · R-466
 - Broadcast() - wLUDP method · R-466
 - ChangeDir() - wLFTP method · R-403
 - ChFileMod() - wLFTP method · R-403
 - ChMod() - wLFTP method · R-404
 - ClearAll() - wLCOOKIE method · R-40
 - ClearDNSCache() - wLHTTP method (protocol mode) · R-41
 - ClearSSLCache() - wLHTTP method (protocol mode) · R-42
 - Close() - wLOUTPUTFILE method · R-45
 - CloseConnection() - wLHTTP method (protocol mode) · R-46
 - Compare() - TABLECOMPARE method · R-49, P-149
 - Connect()
 - wLIMAP method · R-415

- wINNTTP method · R-429
- wIPOP method · R-438
- wISMTTP method · R-447
- wITCP method · R-453
- wITelnet method · R-459
- CreateMailbox() - wIIMAP method · R-416
- Delete()
 - wICookie method · R-70
 - wIFTP method · R-404
 - wIIMAP method · R-416
 - wIPOP method · R-439
- delete() - wIOutputFile method · R-70
- DeleteAttachment()
 - wINNTTP method · R-430
 - wISMTTP method · R-448
- DeleteFile() - wIFTP method · R-404
- DeleteMailbox() - wIIMAP method · R-416
- Dir() - wIFTP method · R-405
- Disconnect()
 - wIIMAP method · R-416
 - wINNTTP method · R-430
 - wIPOP method · R-439
 - wISMTTP method · R-448
 - wITCP method · R-454
 - wITelnet method · R-459
- Download() - wIFTP method · R-405
- DownloadFile() - wIFTP method · R-405
- Erase()
 - wITCP method · R-454
 - wITelnet method · R-459
 - wIUDP method · R-466
- ExpectNavigation() - wIBrowser method · R-97
- FindObject() - wIBrowser method · R-102
- Forward() - wIBrowser method · R-106
- Get()
 - wIGeneratorGlobal method · R-110
 - wIHttp method (protocol mode) · R-111
 - wISystemGlobal method · R-110
- GetArticle() - wINNTTP method · R-430
- GetArticleCount() - wINNTTP method · R-431
- GetCurrentMessageID() - wIPOP method · R-439
- GetCurrentPath() - wIFTP method · R-406
- GetCurrentValue() - wIHttp method (protocol mode) · R-114
- GetFieldValue() - wIHtml method (protocol mode) · R-115
- GetFieldValueInForm() - wIHtml method (protocol mode) · R-116
- GetFormAction() - wIHtml method (protocol mode) · R-117
- GetFrameByUrl() - wIHtml method (protocol mode) · R-118
- GetFrameUrl() - wIHtml method (protocol mode) · R-119
- GetHeaderValue() - wIHtml method (protocol mode) · R-120
- GetHost() - wIHtml method (protocol mode) · R-121
- GetHostName() - wIHtml method (protocol mode) · R-122
- GetIPAddress() - wIHttp method (protocol mode) · R-123
- GetLinkByName() - wIHtml method (protocol mode) · R-127
- GetLinkByUrl() - wIHtml method (protocol mode) · R-128
- GetMailboxSize() - wIPOP method · R-439
- GetMessage() - wIException method · R-129
- GetMessageCount()
 - wIIMAP method · R-417
 - wIPOP method · R-440
- GetPortNum() - wIHtml method (protocol mode) · R-131
- GetQSFieldValue() - wIHtml method (protocol mode) · R-132
- GetSeverity() - wIException method · R-133
- GetStatusLine()
 - wIFTP method · R-406
 - wIHtml method (protocol mode) · R-134
 - wIIMAP method · R-417

- wlNNTP method · R-431
- wlPOP method · R-440
- GetStatusNumber() - wlHtml method (protocol mode) · R-135
- GetUri() - wlHtml method (protocol mode) · R-136
- GroupOverview() - wlNNTP method · R-431
- Head() - wlHttp method (protocol mode) · R-138
- IdentifyObject() - wlHttp method (protocol mode) · R-147
- ListGroups() - wlNNTP method · R-431
- ListLocalFiles() - wlFTP method · R-406
- ListMailboxes() - wlIMAP method · R-417
- load()
 - wlXmIs method · R-171
 - XML DOM method · P-175
- loadXML()
 - wlXmIs method · R-175
 - XML DOM method · P-175
- Logoff() - wlFTP method · R-407
- Logon() - wlFTP method · R-407
- MakeDir() - wlFTP method · R-407
- Navigate() - wlBrowser method · R-186
- Num() - wlRand method · R-188
- Open() - wlOutputFile method · R-193
- OpenStream() - wlMediaPlayer method · R-194
- Pause() - wlMediaPlayer method · R-200
- Play() - wlMediaPlayer method · R-200
- Post() - wlHttp method (protocol mode) · R-202
- PostArticle() - wlNNTP method · R-432
- Prepare() - TableCompare method · R-205, P-149
- Range() - wlRand method · R-211
- Receive()
 - wlTCP method · R-454
 - wlTelnet method · R-459
 - wlUDP method · R-467
- RecentMessageCount() - wlIMAP method · R-417
- Refresh() - wlBrowser method · R-213
- RemoveDir() - wlFTP method · R-407
- Rename() - wlFTP method · R-408
- RenameMailbox() - wlIMAP method · R-418
- ReportLog() - wlException method · R-215
- Reset()
 - wlOutputFile method · R-219
 - wlPOP method · R-440
- Resume() - wlMediaPlayer method · R-220
- Retrieve()
 - wlIMAP method · R-418
 - wlPOP method · R-440
- Search() - wlIMAP method · R-418
- Seed() - wlRand method · R-231
- Select() - wlRand method · R-232
- Send()
 - wlSMTP method · R-448
 - wlTCP method · R-455
 - wlTelnet method · R-460
 - wlUDP method · R-467
- SendCommand()
 - wlFTP method · R-408
 - wlIMAP method · R-420
 - wlNNTP method · R-432
 - wlPOP method · R-441
 - wlSMTP method · R-448
- Set()
 - wlCookie method · R-238
 - wlGeneratorGlobal method · R-237
 - wlSystemGlobal method · R-237
- setType() - WSCComplexObject method · R-242
- setValue() - WSCComplexObject method · R-243
- SetWindow() - wlBrowser method · R-244
- Stop() - wlMediaPlayer method · R-278
- SubscribeMailbox() - wlIMAP method · R-420
- SyncDOM() - wlBrowser method · R-280

- TableCompare() constructor · R-287
 UnBind() - wlUDP method · R-467
 UnsubscribeMailbox() - wlIMAP method · R-420
 Upload() - wlFTP method · R-408
 UploadFile() - wlFTP method · R-409
 UploadUnique() - wlFTP method · R-409
 Validate() - wlBrowser method · R-309
 Verify() - wlSMTP method · R-449
 wlClear() - wlHttp method (protocol mode) · R-328
 wlClick() - input object method · R-326
 wlClose() - window method · R-329
 wlException() constructor · R-333
 WLFtp() constructor · R-409
 wlGet() - data collections method (protocol mode) · R-335
 wlGetAllForms() - document method (protocol mode) · R-337
 wlGetAllFrames() - document method (protocol mode) · R-337
 wlGetAllLinks() - document method (protocol mode) · R-338
 WLImap() constructor · R-420
 wlMediaPlayer() constructor · R-345
 wlMouseDown() - input object method · R-347
 wlMouseOver() - input object method · R-348
 wlMouseUp() - input object method · R-349
 wlMultiSelect() - input object method · R-350
 WLNntp() constructor · R-433
 wlOutputFile() constructor · R-354
 WLPop() constructor · R-441
 wlSelect() - input object method · R-358
 wlSet() - data collections method (protocol mode) · R-359
 WLSmtp() constructor · R-449
 wlSubmit() - form method · R-362
 WLTop() constructor · R-455
 WLTelnet() constructor · R-460
 wlTypeIn() - input object method · R-367
 WLUDP() constructor · R-468
 WLXmlDocument() - wlXmIs constructor · R-369, P-173, P-174
 Write() - wlOutputFile method · R-372
 Writeln() - wlOutputFile method · R-373
 WSComplexObject() constructor · R-375
 WSWebService() constructor · R-379
 mixed clients · P-70
 MultiIPSupport - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-183
 Multiple IP Support · See MultiIPSupport, wlGlobals property
- ## N
- Name
 element property · R-184
 form property · R-184
 frames property · R-184
 image property · R-184
 link property · R-184
 location property · R-184
 map property · R-184
 window property · R-184
 wlMeta property · R-184
 named HTTP transactions
 using ExpectNavigation() (protocol mode) · P-139
 using Get() and Post() (protocol mode) · R-112, R-203, P-139
 named navigation activities - using ExpectNavigation() · R-97
 Native Browsing (XML) · P-173
 Navigate() - wlBrowser method · R-186
 navigation blocks · R-13, P-18, P-65
 NextPrompt
 wlTCP property · R-452
 wlTelnet property · R-457
 NextSize - wlTCP property · R-452

NNTP
 Network News Transfer Protocol · R-425
 Sample Code · R-433
 nodes - in an Agenda Tree · P-64
 Normal Client Type · P-104
 normal execution sequence · P-67
 NT Challenge Response
 authentication (protocol mode) · R-187
 NTPassword - wlGlobals, wlLocals, wlHttp
 property (protocol mode) · R-187
 NTUserName - wlGlobals, wlLocals, wlHttp
 property (protocol mode) · R-187
 Num() - wlRand method · R-188
 NumOfResponses - wlUDP property · R-464

O

ObjectProperty - wlBrowser property · R-189
 objects
 area · R-24
 Button · R-34
 cell (protocol mode) · R-35
 Checkbox · R-38
 creating · R-353, R-354, P-85
 Div · R-77
 document · R-79
 element · R-83
 File · R-99
 form · R-103
 global · P-118, P-119
 image · R-148
 InputButton · R-157
 InputCheckbox · R-158
 InputFile · R-159
 InputImage · R-160
 InputPassword · R-161
 InputRadio · R-163
 InputText · R-164
 limited · P-115
 LineArray · P-86
 link · R-169
 local · P-116
 map · R-178
 option · R-195
 Radiobutton · R-210
 Reset · R-218
 row (protocol mode) · R-223
 Select · R-232
 Span · R-254
 Submit · R-279
 table (protocol mode) · R-284
 TableCell (protocol mode) · R-285
 TableCompare (protocol mode) · R-286, P-145
 Text · R-291
 TextArea · R-292
 title · R-296
 UIContainer · R-301
 window · R-324
 wlBrowser · R-325
 wlCookie · R-330, P-238
 wlException · R-332, P-109
 wlFTP · R-400
 wlGeneratorGlobal · R-334, P-123
 wlGlobals (protocol mode) · R-339, P-119
 wlHeader (protocol mode) · R-340
 wlHtml (protocol mode) · R-342
 wlHttp (protocol mode) · R-343, P-115
 wlIMAP · R-413
 wlLocals (protocol mode) · R-343, P-117
 wlMediaPlayer · R-344
 wlMeta (protocol mode) · R-346
 wlNNTP · R-425
 wlOutputFile · R-351, P-83
 wlPOP · R-436
 wlRand · R-355
 wlSearchPair · R-356
 wlSMTP · R-444
 wlSystemGlobal · R-363, P-125

- wlTable (protocol mode) · R-364
 - wlTCP · R-451
 - wlTelnet · R-457
 - wlUDP · R-463
 - wlXmIs · R-370
 - WSComplexObject · R-374
 - WSWebService · R-378
 - Ok - wlBrowser property (dialog box) · R-75
 - OkCancel - wlBrowser property (dialog box) · R-75
 - OnClick
 - Button property · R-190
 - Div property · R-190
 - link property · R-190
 - script property · R-190
 - Span property · R-190
 - TableCell property · R-190
 - UIContainer property · R-190
 - OnErrorTerminateAgenda() function · P-111
 - OnErrorTerminateClient() function · P-111
 - OnMouseOver
 - Div property · R-192
 - link property · R-192
 - Span property · R-192
 - TableCell property · R-192
 - UIContainer property · R-192
 - OnScriptAbort() function · P-110, P-111
 - Open()
 - function · R-124, R-193, P-87
 - wlOutputFile method · R-193
 - opening external files · R-193
 - OpenStream() - wlMediaPlayer method · R-194
 - option object · R-195
 - example (protocol mode) · P-252
 - properties
 - defaultselected · R-69
 - selected · R-233
 - Text · R-291
 - value · R-310
 - options
 - collection · R-195
 - element property · R-195
 - Organization - wlNNTP property · R-427
 - OutBufferSize
 - wlTCP property · R-452
 - wlUDP property · R-464
 - OuterLink - image property · R-197
 - Outfile
 - wlBrowser property · R-197
 - wlFTP property · R-401
 - wlIMAP property · R-414
 - wlNNTP property · R-427
 - wlPOP property · R-437
 - wlTCP property · R-452
 - wlTelnet property · R-457
 - wlUDP property · R-464
 - output files · R-351, P-83
 - functions · R-100
 - Close() · R-45
 - Open() · R-193
 - overhead, effect on statistics · P-144
- P
- Page Verification Wizard · P-135
 - Parent Transaction Instance Tree (fourth level) · P-155
 - parsing
 - example
 - HTML (protocol mode) · P-242
 - HTTP header (protocol mode) · R-341, P-255
 - links · R-357
 - example (protocol mode) · P-220
 - files · R-125, P-86
 - tree illustration (protocol mode) · P-221
 - PassiveMode - wlFTP property · R-401
 - PassWord
 - wlFTP property · R-401

- wlGlobals, wlLocals, wlHttp property (protocol mode) · R-198
- wlIMAP property · R-415
- wlNNTP property · R-427
- wlPOP property · R-437
- pathname - link and location property · R-199
- pattern matching
 - with wlHeader (protocol mode) · R-340
 - with wlSearchPair · R-356
- Pause() - wlMediaPlayer method · R-200
- pause, inserting a programmed · R-247
- pausing in mid-session · P-29
- performance monitors · P-9
- persistent connection (protocol mode) · R-165
- Play() - wlMediaPlayer method · R-200
- POP
 - Post Office Protocol · R-436
 - Sample Code · R-441
- port - link and location property · R-202
- Post()
 - with named transactions (protocol mode) · R-203
 - wlHttp method (protocol mode) · R-202
- PostArticle() - wlNNTP method · R-432
- Prepare() - TableCompare method · R-205
- Probing Clients · P-9
- ProbingClientThreads - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-207
- programming Agendas
 - advanced features · P-161
- Prompt - wlBrowser property (dialog box) · R-75
- properties
 - action - form property · R-19
 - AdjacentText - element property · R-23
 - Alert - wlBrowser property (dialog box) · R-75
 - Alt
 - area property · R-24
 - element property · R-24
 - image property · R-24
 - ArticleText - wlNNTP property · R-425
 - Attachments
 - wlNNTP property · R-425
 - wlSMTP property · R-444
 - AttachmentsEncoding
 - wlNNTP property · R-425
 - wlSMTP property · R-444
 - AttachmentsTypes
 - wlNNTP property · R-426
 - wlSMTP property · R-445
 - AutoDelete - wlPOP property · R-436
 - Bcc - wlSMTP property · R-445
 - bitrate - wlMediaPlayer property · R-30
 - browser configuration
 - property set for wlGlobals, wlHttp, wlLocals · R-31
 - search order precedence · P-127
 - BrowserCache - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-33
 - Button - document property · R-34
 - Cc - wlSMTP property · R-445
 - cellIndex - cell property · R-37
 - cells
 - row property · R-35
 - wlTable property · R-35
 - Checkbox - document property · R-38
 - checked - element property · R-40
 - ClientAuthentication - wlBrowser property (dialog box) · R-75
 - ClientCertificate - wlBrowser property (dialog box) · R-75
 - cols
 - element property · R-48
 - wlTable property · R-48
 - CompareColumns - TableCompare property · R-51, P-148
 - CompareRows - TableCompare property · R-52, P-147
 - Confirm - wlBrowser property (dialog box) · R-75
 - connectionBandwidth - wlMediaPlayer property · R-53

- ConnectionSpeed - wIGlobals property (protocol mode) · R-54
- ContainerTag - UIContainer property · R-55
- ContainingMap - area property · R-56
- content - wIMeta property · R-56
- Cookie - wIBrowser property (dialog box) · R-75
- Coords - area property · R-57
- CurrentMessage - wIIMAP property · R-413
- CurrentMessageID - wIIMAP property · R-413
- currentPosition - wIMediaPlayer property · R-62
- currentStreamName - wIMediaPlayer property · R-63
- Data
 - wIFTP property · R-400
 - wIHttp property (protocol mode) · R-63
- Data - wIHttp property (protocol mode) · P-232
- DataFile
 - wIFTP property · R-400
 - wIHttp property (protocol mode) · R-65
- DataFile - wIHttp property (protocol mode) · P-233
- DefaultAuthentication - wIGlobals, wILocals, wIHttp property (protocol mode) · R-66
- defaultchecked - element property · R-68
- defaultselected - option property · R-69
- defaultvalue - element property · R-69
- Details - TableCompare property · R-72, P-146
- Dialog box property set for wIBrowser · R-73
- DisableSleep - wIBrowser property · R-75
- Div - document property · R-77
- DNSUseCache - wIGlobals, wILocals, wIHttp property (protocol mode) · R-78
- document
 - window property · R-79
 - wIFTP property · R-400
 - wIIMAP property · R-413
 - wINNTTP property · R-426
 - wIPOP property · R-436
 - wITCP property · R-451
 - wITelnet property · R-457
 - wIUDP property · R-463
- duration - wIMediaPlayer property · R-80
- elements - form property · R-83
- encoding - form property · R-85
- Erase - wIHttp property (protocol mode) · R-87, P-116, P-234
- event
 - script property · R-92
 - TableCell property · R-92
 - UIContainer property · R-92
- ExceptionEnabled - wIBrowser property · R-92
- ExpectedDocument - wIHttp property (protocol mode) · R-93
- ExpectedDOMID - wIHttp property (protocol mode) · R-94
- ExpectedID - wIHttp property (protocol mode) · R-95
- ExpectedLocation - wIHttp property (protocol mode) · R-95
- ExpectedName - wIHttp property (protocol mode) · R-96
- ExpectedText - wIHttp property (protocol mode) · R-97
- File - document property · R-99
- FileName - wIHttp.DataFile property · R-101
- fileName - wIMediaPlayer property · R-101
- forms - document property · R-103
- frames
 - document property · R-107
 - window property · R-107
- From
 - wINNTTP property · R-427
 - wISMTTP property · R-445
- FtpAuthentication - wIBrowser property (dialog box) · R-75
- Group - wINNTTP property · R-427

- hash - link and location property · R-137
- Header - wHttp property (protocol mode) · R-138, P-233
- Headers[] - wPOP property · R-437
- HistoryLimit - wGlobals property · R-140
- host - link and location property · R-141
- hostname - link and location property · R-141
- href - link and location property · R-142
- httpEquiv - wMeta property · R-143
- id
 - area property · R-144
 - element property · R-144
 - form property · R-144
 - frames property · R-144
 - image property · R-144
 - link property · R-144
 - location property · R-144
 - map property · R-144
 - script property · R-144
 - table property · R-144
 - TableCell property · R-144
 - UIContainer property · R-144
 - wTable property · R-144
 - wXmIs property · R-144
- images - document property · R-148
- InBufferSize
 - wTCP property · R-451
 - wUDP property · R-463
- index - frames property · R-151
- innerHTML
 - script property · R-153
 - wXmIs property · R-153
- InnerHTML - cell property · R-153
- InnerImage
 - button property · R-154
 - element property · R-154
 - link property · R-154
 - location property · R-154
 - TableCell property · R-154
 - UIContainer property · R-154
- InnerLink - image property · R-155
- InnerText
 - Button property · R-156
 - cell property · R-156
 - Div property · R-156
 - element property · R-156
 - link property · R-156
 - location property · R-156
 - Span property · R-156
 - TableCell property · R-156
 - UIContainer property · R-156
- InputButton - document property · R-157
- InputCheckbox - document property · R-158
- InputFile - document property · R-159
- InputImage - document property · R-160
- InputPassword - document property · R-161
- InputRadio - document property · R-163
- InputText - document property · R-164
- KeepAlive - wGlobals, wLocals, wHttp property (protocol mode) · R-165
- key
 - Header property · R-166
 - wHeader property · R-166
 - wSearchPair property · R-166
- language - script property · R-167
- length - collection property · R-167
- links - document property · R-169
- LoadGeneratorThreads - wGlobals, wLocals, wHttp property (protocol mode) · R-173
- LocalHost - wUDP property · R-463
- LocalPort
 - wTCP property · R-451
 - wUDP property · R-464
- location
 - document property · R-176
 - window property · R-176
- Mailbox - wIMAP property · R-414
- MatchBy - TableCompare property · R-178, P-147
- MaxDatagramSize - wUDP property · R-464

- MaxHeadersLength - wINNTP property · R-427
- MaxLength - element property · R-180
- MaxLines
 - wIIMAP property · R-414
 - wIPOP property · R-437
- Message - wSMTP property · R-446
- method - form property · R-182
- MultiIPSupport - wIGlobals, wILocals, wIHttp property (protocol mode) · R-183
- Name
 - element property · R-184
 - form property · R-184
 - frames property · R-184
 - image property · R-184
 - link property · R-184
 - location property · R-184
 - map property · R-184
 - window property · R-184
 - wIMeta property · R-184
- NextPrompt
 - wITCP property · R-452
 - wITelnet property · R-457
- NextSize - wITCP property · R-452
- NTPassWord - wIGlobals, wILocals, wIHttp property (protocol mode) · R-187
- NTUserName - wIGlobals, wILocals, wIHttp property (protocol mode) · R-187
- NumOfResponses - wIUDP property · R-464
- ObjectProperty - wIBrowser property · R-189
- Ok - wIBrowser property (dialog box) · R-75
- OkCancel - wIBrowser property (dialog box) · R-75
- OnClick
 - Button property · R-190
 - Div property · R-190
 - link property · R-190
 - script property · R-190
 - Span property · R-190
 - TableCell property · R-190
 - UIContainer property · R-190
- OnMouseOver
 - Div property · R-192
 - link property · R-192
 - Span property · R-192
 - TableCell property · R-192
 - UIContainer property · R-192
- options - element property · R-195
- Organization - wINNTP property · R-427
- OutBufferSize
 - wITCP property · R-452
 - wIUDP property · R-464
- OuterLink - image property · R-197
- Outfile
 - wIBrowser property · R-197
 - wIFTP property · R-401
 - wIIMAP property · R-414
 - wINNTP property · R-427
 - wIPOP property · R-437
 - wITCP property · R-452
 - wITelnet property · R-457
 - wIUDP property · R-464
- PassiveMode - wIFTP property · R-401
- PassWord
 - wIFTP property · R-401
 - wIGlobals, wILocals, wIHttp property (protocol mode) · R-198
 - wIIMAP property · R-415
 - wINNTP property · R-427
 - wIPOP property · R-437
- pathname - link and location property · R-199
- port - link and location property · R-202
- ProbingClientThreads - wIGlobals, wILocals, wIHttp property (protocol mode) · R-207
- Prompt - wIBrowser property (dialog box) · R-75
- Properties[] - wIBrowser property · R-316
- protocol
 - image property · R-208
 - link and location property · R-208

- Proxy - wGlobals, wLocals, wHttp property (protocol mode) · R-209
- ProxyPassWord - wGlobals, wLocals, wHttp property (protocol mode) · R-209
- ProxyUserName - wGlobals, wLocals, wHttp property (protocol mode) · R-209
- Radiobutton - document property · R-210
- ReceiveMessageText
 - wTCP property · R-452
 - wTelnet property · R-458
 - wUDP property · R-465
- ReceiveTimeout - wGlobals, wLocals, wHttp property (protocol mode) · R-212
- RedirectionLimit - wBrowser property · R-212
- References - wNNTP property · R-428
- ReplyTo
 - wNNTP property · R-428
 - wSMTP property · R-446
- ReportUnexpectedRows - TableCompare property · R-216, P-147
- RequestedPackets - wUDP property · R-465
- RequestRetries - wGlobals, wLocals, wHttp property (protocol mode) · R-217
- Reset - document property · R-218
- rowIndex - row property · R-225
- rows
 - elements property · R-223
 - wTable property · R-223
- SaveSource - wGlobals, wLocals, wHttp property (protocol mode) · R-226
- SaveTransaction - wGlobals property (protocol mode) · R-227
- SaveWSTransaction - wGlobals property (protocol mode) · R-228
- scripts - document property · R-229
- search - link and location property · R-230
- Select - document property · R-232
- selected - option property · R-233
- selectedIndex
 - element property · R-234
 - location property · R-234
- Shape - area property · R-246
- Size
 - element property · R-247
 - wFTP property · R-401
 - wIMAP property · R-415
 - wNNTP property · R-428
 - wPOP property · R-438
 - wSMTP property · R-446
 - wTCP property · R-453
 - wTelnet property · R-458
 - wUDP property · R-465
- SleepDeviation - wBrowser property · R-250
- SleepRandomMax - wBrowser property · R-251
- SleepRandomMin - wBrowser property · R-253
- Span - document property · R-254
- src
 - image property · R-255
 - script property · R-255
 - wXmL property · R-255
- SSLBitLimit - wGlobals property (protocol mode) · R-257
- SSLClientCertificateFile - wGlobals, wLocals, wHttp property (protocol mode) · R-260
- SSLClientCertificatePassword - wGlobals, wLocals, wHttp property (protocol mode) · R-260
- SSLCryptoStrength - wGlobals property (protocol mode) · R-262
- SSLUseCache - wGlobals, wLocals, wHttp property (protocol mode) · R-274
- SSLVersion - wGlobals, wLocals, wHttp property (protocol mode) · R-275
- StartByte - wFTP property · R-401
- state - wMediaPlayer property · R-277
- string - title property · R-279
- Subject
 - wNNTP property · R-428
 - wSMTP property · R-446

- Submit - document property · R-279
- tagName - cell property · R-289
- target
 - form property · R-290
 - link property · R-290
- Text
 - document property · R-291
 - option property · R-291
- TextArea - document property · R-292
- Timeout
 - wlTCP property · R-453
 - wlTelnet property · R-458
 - wlUDP property · R-465
- TimeoutSeverity - wlBrowser property · R-293
- title
 - Button property · R-296
 - Div property · R-296
 - document property · R-296
 - element property · R-296
 - frames property · R-296
 - image property · R-296
 - InputButton property · R-296
 - InputCheckbox property · R-296
 - link property · R-296
 - location property · R-296
 - scripts property · R-296
 - Span property · R-296
 - TableCell property · R-296
 - UIContainer property · R-296
 - window property · R-296
- To
 - wlNNTP property · R-428
 - wlSMTP property · R-446
- TransactionTime - wlBrowser property · R-298
- TransferMode - wlFTP property · R-402
- type
 - element property · R-300
 - wlMediaPlayer property · R-300
- Type
 - wlBrowser property · R-300
 - wlHttp.Data property · R-300
 - wlHttp.DataFile property · R-300
 - wlSMTP property · R-446
- URL
 - area property · R-303
 - element property · R-303
 - form property · R-303
 - frames property · R-303
 - image property · R-303
 - link property · R-303
 - location property · R-303
 - window property · R-303
 - wlMeta property · R-303
- Url - wlGlobals property (protocol mode) · R-303
- UseHistory - wlGlobals property · R-304
- UserAgent - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-306
- user-defined
 - wlBrowser properties · R-305
 - wlGlobals properties · R-305
- UserName
 - wlFTP property · R-402
 - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-307
 - wlIMAP property · R-415
 - wlNNTP property · R-429
 - wlPOP property · R-438
- UsingTimer - wlHttp property (protocol mode) · R-308
- value
 - element property · R-310
 - InputButton property · R-310
 - InputCheckbox property · R-310
 - InputRadio property · R-310
 - option property · R-310
 - wlHeader property · R-310
 - wlSearchPair property · R-310
- Value - wlHttp.Data property · R-311

Verification - wIBrowser property · R-313, R-316
 verification test property set for wIBrowser · R-315
 Version - wIGlobals, wILocals, wIHttp property (protocol mode) · R-322
 window (frames collection) · R-324
 wIHeaders - document property (protocol mode) · R-340
 wIMetas - document property (protocol mode) · R-346
 wISearchPairs - link and location property · R-357
 wISource
 document property (protocol mode) · R-361
 wIPOP property · R-438
 wIStatusLine - document property (protocol mode) · R-361
 wIStatusNumber - document property (protocol mode) · R-362
 wITables - document property (protocol mode) · R-364
 wITarget - wIHttp property (protocol mode) · R-366
 wIVersion - document property (protocol mode) · R-368
 wIXmls - document property · R-370
 XMLDocument - wIXmls property · R-380
 YesNo - wIBrowser property (dialog box) · R-75
 YesNoCancel - wIBrowser property (dialog box) · R-75
 Properties[] - wIBrowser property · R-316
 protocol
 image property · R-208
 link and location property · R-208
 Proxy - wIGlobals, wILocals, wIHttp property (protocol mode) · R-209
 ProxyPassWord - wIGlobals, wILocals, wIHttp property (protocol mode) · R-209
 ProxyUserName - wIGlobals, wILocals, wIHttp property (protocol mode) · R-209

Q

Quoted flag · R-425, R-429, R-444, R-447

R

Radiobutton
 document property · R-210
 object · R-210
 random number generator · R-355
 example · R-355, P-242
 Range() - wIRand method · R-211
 reading input files, example · P-89
 Receive()
 wITCP method · R-454
 wITelnet method · R-459
 wIUDP method · R-467
 ReceiveMessageText
 wITCP property · R-452
 wITelnet property · R-458
 wIUDP property · R-465
 ReceiveTimeout - wIGlobals, wILocals, wIHttp property (protocol mode) · R-212
 RecentMessageCount() - wIIMAP method · R-417
 redirection - detecting (protocol mode) · P-251
 RedirectionLimit - wIBrowser property · R-212
 References - wI NNTP property · R-428
 Refresh() - wIBrowser method · R-213
 remote ActiveX object access · P-203
 Remote Data Service (RDS) support · P-203
 RemoveDir() - wI FTP method · R-407
 Rename() - wI FTP method · R-408
 RenameMailbox() - wIIMAP method · R-418
 ReplyTo
 wI NNTP property · R-428
 wI SMTP property · R-446
 ReportEvent() function · R-214
 reporting events · R-214

ReportLog() - wlException method · R-215
 reports, for functional testing and transaction verification · P-151
 ReportUnexpectedRows - TableCompare property · R-216
 RequestedPackets - wlUDP property · R-465
 RequestRetries - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-217
 Reset
 document property · R-218
 object · R-218
 Reset()
 wlOutputFile method · R-219
 wlPOP method · R-440
 Resume() - wlMediaPlayer method · R-220
 Retrieve()
 wlIMAP method · R-418
 wlPOP method · R-440
 return codes
 synchronization points · P-49
 transactions · P-52, P-108
 RoundNum variable · P-57, R-221
 rounds
 aborting · P-110
 cleanup at end · P-69
 example, identifying · P-58, R-146
 identification number of current round · P-57, R-221
 row object
 (protocol mode) · R-223
 properties
 cells · R-35
 rowIndex · R-225
 rowIndex - row property · R-225
 rows
 elements property · R-223
 wlTable property · R-223
 rules of scope · P-114

S

Sample Code
 FTP · R-410
 IMAP · R-421
 NNTP · R-433
 POP · R-441
 SMTP · R-449
 TCP · R-456
 Telnet · R-460
 UDP · R-468
 SaveSource - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-226
 SaveTransaction - wlGlobals property (protocol mode) · R-227
 SaveWSTransaction - wlGlobals property (protocol mode) · R-228
 scheduled clients · P-69
 scope rules · P-114
 user-defined variables · P-114
 script object
 properties
 event · R-92
 id · R-144
 innerHTML · R-153
 language · R-167
 OnClick · R-190
 src · R-255
 title · R-296
 scripts
 collection · R-229
 document property · R-229
 search - link and location property · R-230
 search order precedence
 browser configuration properties · P-127
 copy file locations · P-82
 include file locations · P-80
 Search() - wlIMAP method · R-418
 security
 during Web access · R-4

- example, connecting to a secure internet server (protocol mode) · P-254
- for data transmission · P-94
- for Web applications · P-91
- function list · R-256
- Secure HTTP (protocol mode) · R-275
- secure internet server, connection example (protocol mode) · P-254
- SSLCipherSuiteCommand() function · R-258
- SSLDisableCipherID() function · R-263
- SSLDisableCipherName() function · R-265
- SSLEnableCipherID() function · R-266
- SSLEnableCipherName() function · R-267
- SSLGetCipherCount() function · R-268
- SSLGetCipherID() function · R-269
- SSLGetCipherInfo() function · R-270
- SSLGetCipherName() function · R-271
- SSLGetCipherStrength() function · R-272
- Seed() - wRand method · R-231
- Select
 - document property · R-232
 - object · R-232
- Select() - wRand method · R-232
- selected - option property · R-233
- selectedIndex
 - element property · R-234
 - location property · R-234
- Send()
 - wSMTP method · R-448
 - wTCP method · R-455
 - wTelnet method · R-460
 - wUDP method · R-467
- SendCommand()
 - wFTP method · R-408
 - wIMAP method · R-420
 - wNNTP method · R-432
 - wPOP method · R-441
 - wSMTP method · R-448
- SendCounter() function · R-234
- SendMeasurement() function · R-235
- SendTimer() function · R-236
- sequence of events in Agenda execution · P-67
- session ID
 - example comparing ASM and MSM (protocol mode) · P-246
 - example using Automatic State Management (ASM) (protocol mode) · P-244
 - identifying with Automatic State Management (ASM) (protocol mode) · P-225
- Set()
 - wCookie method · R-238
 - wGeneratorGlobal method · R-237
 - wSystemGlobal method · R-237
- SetFailureReason() function · R-240, P-142
- SetTimer() function · R-241
- setType() - WSCOMPLEXOBJECT method · R-242
- setValue() - WSCOMPLEXOBJECT method · R-243
- SetWindow() - wBrowser method · R-244
- SevereErrorMessage() function · R-245
- Shape - area property · R-246
- ShowAll flag · R-258
- ShowEnabled flag · R-258
- Size
 - element property · R-247
 - wFTP property · R-401
 - wIMAP property · R-415
 - wNNTP property · R-428
 - wPOP property · R-438
 - wSMTP property · R-446
 - wTCP property · R-453
 - wTelnet property · R-458
 - wUDP property · R-465
- sleep modes - setting through wBrowser · R-75, R-250, R-251, R-253
- Sleep() function · P-29, R-247
- SleepDeviation - wBrowser property · R-250
- SleepRandomMax - wBrowser property · R-251
- SleepRandomMin - wBrowser property · R-253
- SMTP
 - Mail Transfer Protocol · R-444
 - Sample Code · R-449

- snapshots · R-280
 - Span
 - document property · R-254
 - object · R-254
 - properties
 - InnerText · R-156
 - OnClick · R-190
 - OnMouseOver · R-192
 - title · R-296
 - src
 - image property · R-255
 - script property · R-255
 - wlXmIs property · R-255
 - SSL
 - example, connecting to server using SSL (protocol mode) · P-253
 - supported protocols · P-94
 - supported protocols (protocol mode) · R-275
 - SSL Cipher Command Suite · R-256
 - SSL_AllCrypto flag · R-262
 - SSL_ExportCryptoOnly flag · R-262
 - SSL_ServerGatedCrypto flag · R-262
 - SSL_StrongCryptoOnly flag · R-262
 - SSLBitLimit - wlGlobals property (protocol mode) · R-257
 - SSLCipherSuiteCommand() function · R-258
 - SSLClientCertificateFile - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-260
 - SSLClientCertificatePassword - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-260
 - SSLCryptoStrength - wlGlobals property (protocol mode) · R-262
 - SSLDisableCipherID() function · R-263
 - SSLDisableCipherName() function · R-265
 - SSLEnableCipherID() function · R-266
 - SSLEnableCipherName() function · R-267
 - SSLGetCipherCount() function · R-268
 - SSLGetCipherID() function · R-269
 - SSLGetCipherInfo() function · R-270
 - SSLGetCipherName() function · R-271
 - SSLGetCipherStrength() function · R-272
 - SSLUseCache - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-274
 - SSLVersion - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-275
 - StartByte - wlFTP property · R-401
 - state - wlMediaPlayer property · R-277
 - statistics
 - effect of overhead · P-144
 - ensuring accuracy · P-144
 - timer functions · P-27, R-295
 - use of timers · P-144
 - Stop() - wlMediaPlayer method · R-278
 - string - title property · R-279
 - Subject
 - wlNNTP property · R-428
 - wlSMTP property · R-446
 - Submit
 - document property · R-279
 - object · R-279
 - SubscribeMailbox() - wlIMAP method · R-420
 - SyncDOM() - wlBrowser method · R-280
 - synchronization points
 - example · R-282
 - return codes · P-49
 - SynchronizationPoint() function · P-48, R-281
 - SynchronizationPoint() function · P-48, R-281
- T**
- Table Compare Definition Wizard · P-145
 - table object
 - (protocol mode) · R-284
 - properties
 - id · R-144
 - TableCell object
 - (protocol mode) · R-285
 - properties
 - event · R-92

- id · R-144
- InnerImage · R-154
- InnerText · R-156
- OnClick · R-190
- OnMouseOver · R-192
- title · R-296
- TableCompare
 - object
 - (protocol mode) · R-286, P-145
 - example using TableCompare()
 - constructor (protocol mode) · P-249
 - methods
 - Compare() · R-49, P-149
 - Prepare() · R-205, P-149
 - TableCompare() constructor · R-287
 - properties
 - CompareColumns · R-51, P-148
 - CompareRows · R-52, P-147
 - Details · R-72, P-146
 - MatchBy · R-178, P-147
 - ReportUnexpectedRows · R-216, P-147
 - Results (WebLOAD Viewer) · P-160
 - Results Tree (three levels) · P-157
 - TableCompare() constructor · R-287
- tables
 - CreateTable() function · R-61
 - expected · R-61
- tagName - cell property · R-289
- target
 - form property · R-290
 - link property · R-290
- target frame (ASM) (protocol mode) · P-225
- TCP
 - Sample Code · R-456
 - Transfer Control Protocol · R-451
- Telnet · R-457
 - Sample Code · R-460
- TerminateAgenda() function · P-68
- TerminateClient() function · P-68
- termination functions · P-66
- Text
 - document property · R-291
 - object · R-291
 - option property · R-291
- text files · R-125, P-86
- TextArea
 - document property · R-292
 - object · R-292
- Thick Client Type · P-104
- Thin Client Type · P-104
- ThreadNum variable · P-57, R-43
- threads · P-68
 - example, identifying clients · P-58, R-146
 - extra, for nested downloads (protocol mode)
 - R-173, R-207
 - unique client number · P-57, R-43
- Timeout
 - wlTCP property · R-453
 - wlTelnet property · R-458
 - wlUDP property · R-465
- TimeoutSeverity - wlBrowser property · R-293
- timers · P-144
 - automatic, for Java class methods · P-187
 - complete function list · R-295
 - example · R-295, P-242
 - SendCounter() function · R-234
 - SendMeasurement() function · R-235
 - SendTimer() function · R-236
 - SetTimer() function · R-241
 - Sleep() function · R-247
 - zeroing · R-241
- title
 - Button property · R-296
 - Div property · R-296
 - document property · R-296
 - element property · R-296
 - frames property · R-296
 - image property · R-296
 - InputButton property · R-296
 - InputCheckbox property · R-296

- link property · R-296
 - location property · R-296
 - object · R-296
 - properties
 - string · R-279
 - scripts property · R-296
 - Span property · R-296
 - TableCell property · R-296
 - UIContainer property · R-296
 - window property · R-296
 - TLS 1.0 protocol · P-94
 - To
 - wlNNTP property · R-428
 - wlSMTP property · R-446
 - tracking reasons for transaction failure · R-240
 - Transaction Failure Reason Grid (second level) · P-153
 - Transaction Grid (first level) · P-152
 - Transaction Instance Grid (third level) · P-154
 - transaction verification
 - BeginTransaction() function · R-28, P-141
 - complete function list · R-297
 - CreateDOM() function · R-59, P-150
 - CreateTable() function · R-61
 - creating an expected DOM · P-150
 - Data Drilling reports · P-151
 - EndTransaction() function · R-85, P-141
 - example using TableCompare() constructor (protocol mode) · P-249
 - named HTTP transactions (protocol mode) · R-112, R-203, P-139
 - named navigation activities · R-97
 - ReportEvent() function · R-214
 - SetFailureReason() function · R-240, P-142
 - TableCompare object · P-145
 - VerificationFunction() (user-defined function) · R-320, P-141
 - transactions
 - adding or defining · P-50
 - HTTP configuration, setting & erasing (protocol mode) · P-116
 - return codes · P-52, P-108
 - TransactionTime - wlBrowser property · R-298
 - TransferMode - wlFTP property · R-402
 - travel agency example (protocol mode) · P-240
 - type
 - element property · R-300
 - wlMediaPlayer property · R-300
 - Type
 - wlBrowser property · R-300
 - wlHttp.Data property · R-300
 - wlHttp.DataFile property · R-300
 - wlSMTP property · R-446
- ## U
- UDP
 - Sample Code · R-468
 - User Datagram Protocol · R-463
 - UIContainer object · R-301
 - properties
 - ContainerTag · R-55
 - event · R-92
 - id · R-144
 - InnerImage · R-154
 - InnerText · R-156
 - OnClick · R-190
 - OnMouseOver · R-192
 - title · R-296
 - UnBind() - wlUDP method · R-467
 - UnsubscribeMailbox() - wlIMAP method · R-420
 - Upload() - wlFTP method · R-408
 - UploadFile() - wlFTP method · R-409
 - UploadUnique() - wlFTP method · R-409
 - URL
 - area property · R-303
 - element property · R-303
 - form property · R-303
 - frames property · R-303
 - image property · R-303

- link property · R-303
- location property · R-303
- window property · R-303
- wlMeta property · R-303
- Url - wlGlobals property (protocol mode) · R-303
- UseHistory - wlGlobals property · R-304
- UserAgent - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-306
- user-defined
 - global properties
 - wlBrowser properties · R-305
 - wlGlobals properties · R-305
 - global variables · P-118
 - limited variables · P-115
 - local variables · P-116
- user-defined variables
 - global · P-121, P-123, P-125
 - rules of scope · P-114
- UserName
 - wlFTP property · R-402
 - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-307
 - wlIMAP property · R-415
 - wlNNTP property · R-429
 - wlPOP property · R-438
- UsingTimer - wlHttp property (protocol mode) · R-308

V

- Validate() - wlBrowser method · R-309
- value
 - element property · R-310
 - InputButton property · R-310
 - InputCheckbox property · R-310
 - InputRadio property · R-310
 - option property · R-310
 - wlHeader property · R-310
 - wlSearchPair property · R-310
- Value - wlHttp.Data property · R-311

- variables
 - ClientNum
 - example · P-58, R-146
 - identification · P-56, R-146
 - ClientNum · P-57, R-43
 - RoundNum · P-57, R-221
 - RoundNum
 - example · P-58, R-146
 - user-defined
 - global · P-118, P-121, P-123, P-125
 - limited · P-115
 - local · P-116
 - rules of scope · P-114
- VCUniqueID() function · R-312
- Verification - wlBrowser property · R-313, R-316
- verification of transactions
 - CreateDOM() function · R-59
 - CreateTable() function · R-61
 - named HTTP transactions (protocol mode) · R-112, R-203, P-139
 - named navigation activities · R-97
 - ReportEvent() function · R-214
 - SetFailureReason() function · R-240
 - VerificationFunction() (user-defined function) · R-320, P-141
- verification tests · R-315, R-316
 - wlBrowser property set · R-315
- VerificationFunction() (user-defined function) · R-320, P-141
- Verify() - wlSMTP method · R-449
- Version - wlGlobals, wlLocals, wlHttp property (protocol mode) · R-322
- Virtual Clients · P-9
 - VCUniqueID() function · R-312
- visual set (ASM) (protocol mode) · P-225

W

- WarningMessage() function · R-323

- web page illustration (protocol mode) · P-220
- Web Services · P-31, R-374, R-378, P-207
- WebLOAD Internet Productivity Pack · P-214
 - error handling · P-214
- WebServices
 - functions
 - WSGetSimpleValue() · R-376
- WebServices Wizard · P-31
- window
 - frames property · R-324
 - object · R-324
 - methods
 - wlClose() · R-329
 - properties
 - document · R-79
 - frames · R-107
 - location · R-176
 - Name · R-184
 - title · R-296
 - URL · R-303
- Windows NT Challenge Response protocol
 - authentication (protocol mode) · R-187
- wizards
 - Form Data Wizard · P-36
 - Page Verification Wizard · P-135
 - Table Compare Definition Wizard · P-145
 - WebServices Wizard · P-31
- WLAfterInitAgenda flag · R-90
- WLAfterRound flag · R-90
- WLAfterTerminateAgenda flag · R-90
- WLAfterTerminateClient flag · R-90
- WLAllAgendas flag · R-21, R-110, R-237, P-123, P-125
- WLBeforeInitClient flag · R-90
- WLBeforeRound flag · R-90
- WLBeforeThreadActivation flag · R-90
- wlBrowser object · R-325
 - dialog box
 - properties, complete list · R-73
 - Dynamic Object Recognition (DOR)
 - properties · R-81
- methods
 - AutoNavigate() · R-26
 - Back() · R-27
 - ExpectNavigation() · R-97
 - FindObject() · R-102
 - Forward() · R-106
 - Navigate() · R-186
 - Refresh() · R-213
 - SetWindow() · R-244
 - SyncDOM() · R-280
 - Validate() · R-309
- properties
 - Alert (dialog box) · R-75
 - ClientAuthentication (dialog box) · R-75
 - ClientCertificate (dialog box) · R-75
 - Confirm (dialog box) · R-75
 - Cookie (dialog box) · R-75
 - Dialog box property set · R-73
 - DisableSleep · R-75
 - ExceptionEnabled · R-92
 - FtpAuthentication (dialog box) · R-75
 - ObjectProperty · R-189
 - Ok (dialog box) · R-75
 - OkCancel (dialog box) · R-75
 - Outfile · R-197
 - Prompt (dialog box) · R-75
 - Properties[] · R-316
 - RedirectionLimit · R-212
 - SleepDeviation · R-250
 - SleepRandomMax · R-251
 - SleepRandomMin · R-253
 - TimeoutSeverity · R-293
 - TransactionTime · R-298
 - Type · R-300
 - user-defined · R-305
 - Verification · R-313, R-316
 - verification test property set · R-315
 - YesNo (dialog box) · R-75
 - YesNoCancel (dialog box) · R-75
- verification test properties · R-315

- wlClear() - wlHttp method (protocol mode) · R-328
- wlClick() - input object method · R-326
- wlClose() - window method · R-329
- wlContent-Type - wlHttp.FormData property (protocol mode) · P-231
- wlCookie object · R-330, P-238
 - example · R-331, P-239
 - methods
 - ClearAll() · R-40
 - Delete() · R-70
 - Set() · R-238
- WLCurrentAgenda flag · R-21, R-110, R-237, P-123, P-125
- WLError flag · P-49, P-52, R-86, R-133, R-281, R-294, R-309, R-320, R-333, P-108
- wlException object · R-332, P-109
 - methods
 - GetMessage() · R-129
 - GetSeverity() · R-133
 - ReportLog() · R-215
 - wlException() constructor · R-333
- wlException() constructor · R-333
- WlExecuteScript flag · R-150
- wlFile-Name - wlHttp.FormData property (protocol mode) · P-231
- wlFTP object · R-400
 - methods
 - Append() · R-402
 - AppendFile() · R-403
 - ChangeDir() · R-403
 - ChFileMod() · R-403
 - ChMod() · R-404
 - Delete() · R-404
 - DeleteFile() · R-404
 - Dir() · R-405
 - Download() · R-405
 - DownloadFile() · R-405
 - GetCurrentPath() · R-406
 - GetStatusLine() · R-406
 - ListLocalFiles() · R-406
 - Logoff() · R-407
 - Logon() · R-407
 - MakeDir() · R-407
 - RemoveDir() · R-407
 - Rename() · R-408
 - SendCommand() · R-408
 - Upload() · R-408
 - UploadFile() · R-409
 - UploadUnique() · R-409
 - WLFtp() constructor · R-409
- properties
 - Data · R-400
 - DataFile · R-400
 - document · R-400
 - Outfile · R-401
 - PassiveMode · R-401
 - PassWord · R-401
 - Size · R-401
 - StartByte · R-401
 - TransferMode · R-402
 - UserName · R-402
- WLFtp() constructor · R-409
- wlGeneratorGlobal object · R-334, P-123
 - methods
 - Add() · R-21
 - Get() · R-110
 - Set() · R-237
- wlGet() - data collections method (protocol mode) · R-335
- wlGetAllForms() - document method (protocol mode) · R-337
- wlGetAllFrames() - document method (protocol mode) · R-337
- wlGetAllLinks() - document method (protocol mode) · R-338
- wlGlobals object · P-119
 - (protocol mode) · R-339
 - Browser configuration
 - property set · R-31
 - example

- connecting to secure server (protocol mode) · P-254
- setting configuration properties (protocol mode) · P-253
- properties
 - Browser configuration property set · R-31
 - BrowserCache (protocol mode) · R-33
 - ConnectionSpeed (protocol mode) · R-54
 - DefaultAuthentication (protocol mode) · R-66
 - DNSUseCache (protocol mode) · R-78
 - HistoryLimit · R-140
 - KeepAlive (protocol mode) · R-165
 - LoadGeneratorThreads (protocol mode) · R-173
 - MultiIPSupport (protocol mode) · R-183
 - NTPassWord (protocol mode) · R-187
 - NTUserName (protocol mode) · R-187
 - PassWord (protocol mode) · R-198
 - ProbingClientThreads (protocol mode) · R-207
 - Proxy (protocol mode) · R-209
 - ProxyPassWord (protocol mode) · R-209
 - ProxyUserName (protocol mode) · R-209
 - ReceiveTimeout (protocol mode) · R-212
 - RequestRetries (protocol mode) · R-217
 - SaveSource (protocol mode) · R-226
 - SaveTransaction (protocol mode) · R-227
 - SaveWSTransaction (protocol mode) · R-228
 - SSLBitLimit (protocol mode) · R-257
 - SSLClientCertificateFile (protocol mode) · R-260
 - SSLClientCertificatePassword (protocol mode) · R-260
 - SSLCryptoStrength (protocol mode) · R-262
 - SSLUseCache (protocol mode) · R-274
 - SSLVersion (protocol mode) · R-275
 - Url (protocol mode) · R-303
 - UseHistory · R-304
 - UserAgent (protocol mode) · R-306
 - user-defined · R-305
 - UserName (protocol mode) · R-307
 - Version (protocol mode) · R-322
- wlHeader object
 - (protocol mode) · R-340
 - example (protocol mode) · R-341, P-255
 - properties
 - key · R-166
 - value · R-310
- wlHeaders
 - collection · R-341
 - document property (protocol mode) · R-340
- wlHtml object
 - (protocol mode) · R-342
 - example (protocol mode) · P-223
 - methods
 - GetFieldValue() · R-115
 - GetFieldValueInForm() · R-116
 - GetFormAction() · R-117
 - GetFrameByUrl() · R-118
 - GetFrameUrl() · R-119
 - GetHeaderValue() · R-120
 - GetHost() · R-121
 - GetHostName() · R-122
 - GetLinkByName() · R-127
 - GetLinkByUrl() · R-128
 - GetPortNum() · R-131
 - GetQSFieldValue() · R-132
 - GetStatusLine() · R-134
 - GetStatusNumber() · R-135
 - GetUri() · R-136
- wlHttp object · P-115
 - (protocol mode) · R-343
 - Automatic State Management (ASM)
 - properties (protocol mode) · R-25, P-225
 - Browser configuration
 - property set · R-31

- data submission properties (protocol mode) · P-229
- example
 - connecting with SSL (protocol mode) · P-253
 - setting configuration properties (protocol mode) · P-253
- methods
 - ClearDNSCache() · R-41
 - ClearSSLCache() · R-42
 - CloseConnection() · R-46
 - Get() · R-111
 - GetCurrentValue() · R-114
 - GetIPAddress() · R-123
 - Head() · R-138
 - IdentifyObject() · R-147
 - Post() · R-202
 - wlClear() · R-328
- properties
 - Browser configuration property set · R-31
 - BrowserCache (protocol mode) · R-33
 - Data (protocol mode) · R-63, P-232
 - DataFile (protocol mode) · R-65, P-233
 - DefaultAuthentication (protocol mode) · R-66
 - DNSUseCache (protocol mode) · R-78
 - Erase (protocol mode) · R-87, P-234
 - ExpectedDocument (protocol mode) · R-93
 - ExpectedDOMID (protocol mode) · R-94
 - ExpectedID (protocol mode) · R-95
 - ExpectedLocation (protocol mode) · R-95
 - ExpectedName (protocol mode) · R-96
 - ExpectedText (protocol mode) · R-97
 - FormData
 - missing fields (protocol mode) · P-230
 - using Get() (protocol mode) · R-105, P-229
 - using Post() (protocol mode) · R-105, P-230
 - working with data files (protocol mode) · P-231
 - Header (protocol mode) · R-138, P-233
 - KeepAlive (protocol mode) · R-165
 - LoadGeneratorThreads (protocol mode) · R-173
 - MultiIPSupport (protocol mode) · R-183
 - NTPassWord (protocol mode) · R-187
 - NTUserName (protocol mode) · R-187
 - PassWord (protocol mode) · R-198
 - ProbingClientThreads (protocol mode) · R-207
 - Proxy (protocol mode) · R-209
 - ProxyPassWord (protocol mode) · R-209
 - ProxyUserName (protocol mode) · R-209
 - ReceiveTimeout (protocol mode) · R-212
 - RequestRetries (protocol mode) · R-217
 - SaveSource (protocol mode) · R-226
 - SSLClientCertificateFile (protocol mode) · R-260
 - SSLClientCertificatePassword (protocol mode) · R-260
 - SSLUseCache (protocol mode) · R-274
 - SSLVersion (protocol mode) · R-275
 - UserAgent (protocol mode) · R-306
 - UserName (protocol mode) · R-307
 - UsingTimer (protocol mode) · R-308
 - Version (protocol mode) · R-322
 - wlTarget (protocol mode) · R-366
- wlHttp.Data object
 - properties
 - Type · R-300
 - Value · R-311
- wlHttp.DataFile object
 - properties
 - FileName · R-101
 - Type · R-300
- wlIMAP object · R-413

- methods
 - Connect() · R-415
 - CreateMailbox() · R-416
 - Delete() · R-416
 - DeleteMailbox() · R-416
 - Disconnect() · R-416
 - GetMessageCount() · R-417
 - GetStatusLine() · R-417
 - ListMailboxes() · R-417
 - RecentMessageCount() · R-417
 - RenameMailbox() · R-418
 - Retrieve() · R-418
 - Search() · R-418
 - SendCommand() · R-420
 - SubscribeMailbox() · R-420
 - UnsubscribeMailbox() · R-420
 - WLMailbox() constructor · R-420
- properties
 - CurrentMessage · R-413
 - CurrentMessageID · R-413
 - document · R-413
 - Mailbox · R-414
 - MaxLines · R-414
 - Outfile · R-414
 - PassWord · R-415
 - Size · R-415
 - UserName · R-415
- WLMailbox() constructor · R-420
- WMLoadChanged flag · P-49, R-281
- wlLocals object · P-117
 - (protocol mode) · R-343
 - Browser configuration
 - property set · R-31
 - example, setting configuration properties (protocol mode) · P-253
 - properties
 - Browser configuration property set · R-31
 - BrowserCache (protocol mode) · R-33
 - DefaultAuthentication (protocol mode) · R-66
 - DNSUseCache (protocol mode) · R-78
 - KeepAlive (protocol mode) · R-165
 - LoadGeneratorThreads (protocol mode) · R-173
 - MultiIPSupport (protocol mode) · R-183
 - NTPassWord (protocol mode) · R-187
 - NTUserName (protocol mode) · R-187
 - PassWord (protocol mode) · R-198
 - ProbingClientThreads (protocol mode) · R-207
 - Proxy (protocol mode) · R-209
 - ProxyPassWord (protocol mode) · R-209
 - ProxyUserName (protocol mode) · R-209
 - ReceiveTimeout (protocol mode) · R-212
 - RequestRetries (protocol mode) · R-217
 - SaveSource (protocol mode) · R-226
 - SSLClientCertificateFile (protocol mode) · R-260
 - SSLClientCertificatePassword (protocol mode) · R-260
 - SSLUseCache (protocol mode) · R-274
 - SSLVersion (protocol mode) · R-275
 - UserAgent (protocol mode) · R-306
 - UserName (protocol mode) · R-307
 - Version (protocol mode) · R-322
- wlMediaPlayer object · R-344
 - methods
 - OpenStream() · R-194
 - Pause() · R-200
 - Play() · R-200
 - Resume() · R-220
 - Stop() · R-278
 - wlMediaPlayer() constructor · R-345
 - properties
 - bitrate · R-30
 - connectionBandwidth · R-53
 - currentPosition · R-62
 - currentStreamName · R-63
 - duration · R-80
 - fileName · R-101

- state · R-277
- type · R-300
- wlMediaPlayer() constructor · R-345
- wlMeta object
 - (protocol mode) · R-346
 - properties
 - content · R-56
 - httpEquiv · R-143
 - Name · R-184
 - URL · R-303
- wlMetas
 - collection · R-346
 - document property (protocol mode) · R-346
- WLMinorError flag · P-52, R-86, R-294, R-309, R-320, P-108
- wlMouseDown() - input object method · R-347
- wlMouseOver() - input object method · R-348
- wlMouseUp() - input object method · R-349
- wlMultiSelect() - input object method · R-350
- wlNNTP object · R-425
 - methods
 - AddAttachment() · R-429
 - Connect() · R-429
 - DeleteAttachment() · R-430
 - Disconnect() · R-430
 - GetArticle() · R-430
 - GetArticleCount() · R-431
 - GetStatusLine() · R-431
 - GroupOverview() · R-431
 - ListGroups() · R-431
 - PostArticle() · R-432
 - SendCommand() · R-432
 - WLNntp() constructor · R-433
 - properties
 - ArticleText · R-425
 - Attachments · R-425
 - AttachmentsEncoding · R-425
 - AttachmentsTypes · R-426
 - document · R-426
 - From · R-427
 - Group · R-427
 - MaxHeadersLength · R-427
 - Organization · R-427
 - Outfile · R-427
 - PassWord · R-427
 - References · R-428
 - ReplyTo · R-428
 - Size · R-428
 - Subject · R-428
 - To · R-428
 - UserName · R-429
- WLNntp() constructor · R-433
- WLOnThreadActivation flag · R-90
- wlOutputFile object · R-351, P-83
 - methods
 - Close() · R-45
 - delete() · R-70
 - Open() · R-193
 - Reset() · R-219
 - wlOutputFile() constructor · R-354
 - Write() · R-372
 - Writeln() · R-373
- wlOutputFile() constructor · R-354
- wlPOP object · R-436
 - methods
 - Connect() · R-438
 - Delete() · R-439
 - Disconnect() · R-439
 - GetCurrentMessageID() · R-439
 - GetMailboxSize() · R-439
 - GetMessageCount() · R-440
 - GetStatusLine() · R-440
 - Reset() · R-440
 - Retrieve() · R-440
 - SendCommand() · R-441
 - WLPop() constructor · R-441
 - properties
 - AutoDelete · R-436
 - document · R-436
 - Headers[] · R-437

- MaxLines · R-437
- Outfile · R-437
- PassWord · R-437
- Size · R-438
- UserName · R-438
- wlSource · R-438
- WLPop() constructor · R-441
- wlRand object · R-355
 - example · R-355
 - methods
 - Num() · R-188
 - Range() · R-211
 - Seed() · R-231
 - Select() · R-232
- WLRandom flag · R-124, R-193, P-86
- wlSearchPair object · R-356
 - example · R-357
 - properties
 - key · R-166
 - value · R-310
- wlSearchPairs
 - collection · R-357
 - link and location property · R-357
- wlSelect() - input object method · R-358
- WLSequential flag · R-124, R-193, P-86
- wlSet() - data collections method (protocol mode) · R-359
- WLSevereError flag · P-52, R-86, R-133, R-294, R-309, R-320, R-333, P-108
- wlSMTP object · R-444
 - methods
 - AddAttachment() · R-447
 - Connect() · R-447
 - DeleteAttachment() · R-448
 - Disconnect() · R-448
 - Send() · R-448
 - SendCommand() · R-448
 - Verify() · R-449
 - WLSmtp() constructor · R-449
 - properties
 - Attachments · R-444
 - AttachmentsEncoding · R-444
 - AttachmentsTypes · R-445
 - Bcc · R-445
 - Cc · R-445
 - From · R-445
 - Message · R-446
 - ReplyTo · R-446
 - Size · R-446
 - Subject · R-446
 - To · R-446
 - Type · R-446
- WLSmtp() constructor · R-449
- wlSource
 - document property (protocol mode) · R-361
 - wlPOP property · R-438
- wlStatusLine - document property (protocol mode) · R-361
- wlStatusNumber - document property (protocol mode) · R-362
- wlSubmit() - form method · R-362
- WLSuccess flag · P-49, P-52, R-86, R-281, R-294, R-309, R-320, P-108
- wlSystemGlobal object · R-363, P-125
 - methods
 - Add() · R-21
 - Get() · R-110
 - Set() · R-237
- wlTable object
 - (protocol mode) · R-364
 - properties
 - cells · R-35
 - cols · R-48
 - id · R-144
 - rows · R-223
- wlTables
 - collection · R-364
 - document property (protocol mode) · R-364
- wlTarget - wlHttp property (protocol mode) · R-366
- wlTCP object · R-451
 - methods

- Connect() · R-453
- Disconnect() · R-454
- Erase() · R-454
- Receive() · R-454
- Send() · R-455
- WLTcp() constructor · R-455
- properties
 - document · R-451
 - InBufferSize · R-451
 - LocalPort · R-451
 - NextPrompt · R-452
 - NextSize · R-452
 - OutBufferSize · R-452
 - Outfile · R-452
 - ReceiveMessageText · R-452
 - Size · R-453
 - Timeout · R-453
- WLTcp() constructor · R-455
- wlTelnet object · R-457
 - methods
 - Connect() · R-459
 - Disconnect() · R-459
 - Erase() · R-459
 - Receive() · R-459
 - Send() · R-460
 - WLTelnet() constructor · R-460
 - properties
 - document · R-457
 - NextPrompt · R-457
 - Outfile · R-457
 - ReceiveMessageText · R-458
 - Size · R-458
 - Timeout · R-458
- WLTelnet() constructor · R-460
- WLTimeout flag · P-49, R-281
- wlTypeIn() - input object method · R-367
- wlUDP object · R-463
 - methods
 - Bind() · R-466
 - Broadcast() · R-466
 - Erase() · R-466
 - Receive() · R-467
 - Send() · R-467
 - UnBind() · R-467
 - WLUDP() constructor · R-468
 - properties
 - document · R-463
 - InBufferSize · R-463
 - LocalHost · R-463
 - LocalPort · R-464
 - MaxDatagramSize · R-464
 - NumOfResponses · R-464
 - OutBufferSize · R-464
 - Outfile · R-464
 - ReceiveMessageText · R-465
 - RequestedPackets · R-465
 - Size · R-465
 - Timeout · R-465
- WLUDP() constructor · R-468
- wlVersion - document property (protocol mode) · R-368
- WLXmlDocument() - wlXmles constructor · R-369, P-173, P-174
- wlXmles
 - collection · R-371
 - document property · R-370
 - object · R-370
 - methods
 - load() · R-171
 - loadXML() · R-175
 - WLXmlDocument() constructor · R-369
 - properties
 - id · R-144
 - innerHTML · R-153
 - src · R-255
 - XMLDocument · R-380
- working with session IDs, example using Automatic State Management (ASM) (protocol mode) · P-246
- Write() - wlOutputFile method · R-372

WriteLine() - wlOutputFile method · R-373
 WSComplexObject object · R-374
 methods
 addProperty() · R-22
 setType() · R-242
 setValue() · R-243
 WSComplexObject() constructor · R-375
 WSComplexObject() constructor · R-375
 WSDL file · R-374, R-378
 WSGetSimpleValue()
 function · R-376
 WSWebService object · R-378
 methods
 WSWebService() constructor · R-379
 WSWebService() constructor · R-379

X

XML DOM
 Data Islands · P-163
 DTD in an XML document · P-178

example
 building a complete database · P-176
 Data Island use · P-167
 HTML property use · R-371
 Native Browsing · P-173
 object · P-163
 methods
 load() · P-175
 loadXML() · P-175
 WLXmlDocument() constructor · P-173, P-174
 object construction · P-173
 working with · P-162
 XmlDocument - wlXmIs property · R-380

Y

YesNo - wIBrowser property (dialog box) · R-75
 YesNoCancel - wIBrowser property (dialog box) · R-75

