JavaScript Reference

Version 12.0

RADVIEW

WebLOAD JavaScript Reference Guide

November 2018, RadView Publication Number WL-PRO-0909-JRM84

For product assistance or information, contact:	
Toll free in the US: Fax: World Wide Web:	1-888-RadView +1-908-864-8099 www.RadView.com
North American Headquarters:	**International Headquarters:**
RadView Software Inc. 991 Highway 22 West, Suite 200 Bridgewater, NJ 08807 Email: info@RadView.com Phone: 908-526-7756 Fax: 908-864-8099 Toll Free: 1-888-RadView	RadView Software Ltd. 13 Haamal Street, Park Afek Rosh Haayin 4809249 Israel Email: info@RadView.com Phone: +972-3-915-7060 Fax: +972-3-915-7011

Table of Contents

Chapter 1

Introduction

Welcome to WebLOAD, the premier performance, scalability, and reliability testing solution for internet applications.

WebLOAD is easy to use and delivers maximum testing performance and value. WebLOAD verifies the scalability and integrity of internet applications by generating a load composed of Virtual Clients that simulate real-world traffic. Probing Clients let you refine the testing process by acting as a single user that measures the performance of targeted activities, and provides individual performance statistics of the internet application under load.

This section provides a brief introduction to WebLOAD technical support, including both documentation and online support.

IMPORTANT NOTE: In previous WebLOAD versions, a WebLOAD script was called an "Agenda". From version 12.0, it is referred to simply as a script. Wherever "Agenda" is still displayed, we are referring to the WebLOAD script.

WebLOAD Recorder was formerly referred to as WebLOAD IDE.

WebLOAD Documentation

WebLOAD is supplied with the following documentation:

WebLOAD™ Installation Guide

Instructions for installing WebLOAD and its add-ons.

WebLOAD™ Recorder User Guide

Instructions for recording, editing, and debugging load test

s to be executed by WebLOAD to test your Web-based applications.

WebLOAD™ Console User Guide

A guide to using WebLOAD console, RadView's load/scalability testing tool to easily and efficiently test your Web-based applications. This guide also includes a quick start section containing instructions for getting started quickly with WebLOAD using the RadView Software test site.

WebLOAD™ Analytics User Guide

Instructions on how to use WebLOAD Analytics to analyze data and create custom, informative reports after running a WebLOAD test session.

WebRM™ User Guide

Instructions for managing testing resources with the WebLOAD Resource Manager.

WebLOAD™ Scripting Guide

Complete information on scripting and editing JavaScript scripts for use in WebLOAD and WebLOAD Recorder.

WebLOAD™ JavaScript Reference Guide

Complete reference information on all JavaScript objects, variables, and functions used in WebLOAD and WebLOAD Recorder test scripts.

WebLOAD™ Extensibility SDK

Instructions on how to develop extensions to tailor WebLOAD to specific working environments.

WebLOAD™ Automation Guide

Instructions for automatically running WebLOAD tests and reports from the command line, or by using the WebLOAD plugin for Jenkins.

WebLOAD™ Web Dashboard User Guide

Instructions for using RadView's Web Dashboard to view, analyze and compare load sessions in a web browser, with full control and customization of the display.

WebLOAD™ Cloud User Guide

Instructions for using RadView's WebLOAD Cloud to view, analyze and compare load sessions in a web browser, with full control and customization of the display.

The guides are distributed with the WebLOAD software in online help format. The guides are also supplied as Adobe Acrobat files. View and print these files using the Adobe Acrobat Reader. Install the Reader from the Adobe website http://www.adobe.com.

Typographical Conventions

Before you start using this guide, it is important to understand the terms and typographical conventions used in the documentation.

For more information on specialized terms used in the documentation, see *Glossary* (on page 483).

The following icons appear next to the text to identify special information.

Table 1: Icon Conventions

Icon	Type of Information
	Indicates a note.
	Indicates a feature that is available only as part of a WebLOAD Add-on.

The following kinds of formatting in the text identify special information.

Table 2: Typographical Conventions

Formatting convention	Type of Information
Special Bold	Items you must select, such as menu options, command buttons, or items in a list.
Emphasis	Use to emphasize the importance of a point or for variable expressions such as parameters.
CAPITALS	Names of keys on the keyboard. for example, SHIFT, CTRL, or ALT.
KEY+KEY	Key combinations for which the user must press and hold down one key and then press another, for example, CTRL+P, or ALT+F4.

Where to Get More Information

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

Online 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 Website

The technical support pages on our website contain:

- The option of opening a ticket
- Links to WebLOAD documentation

Technical Support

For technical support in your use of this product, contact:

North American Headquarters	International Headquarters
e-mail: support@RadView.com Phone: 1-888-RadView (1-888-723-8439) (Toll Free) 908-526-7756 Fax: 908-864-8099	e-mail: support@RadView.com Phone: +972-3-915-7060 Fax: +972-3-915-7011

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, as well as the version and build number.

Chapter 2

Introduction to JavaScript scripts

The *WebLOAD JavaScript Reference Guide* 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 scripts.

Note: Most WebLOAD users do not need this level of detail to create effective testing sessions for their website. Scripts are usually recorded and edited using WebLOAD Recorder, a simple, intuitive interface that provides users with a comprehensive set of testing tools literally at their fingertips, through point-and-click or drag-and-drop convenience. The details in this guide are provided for the convenience of more sophisticated programmers, who may wish to add specific, perhaps complex tailoring to their recorded scripts.

This chapter provides a general introduction to JavaScript scripts.

What are scripts?

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

Scripts are created using WebLOAD Recorder. WebLOAD Recorder 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 the mouse), WebLOAD Recorder records all user actions in a script. During later website testing sessions, WebLOAD simulates every action of the original user and automatically handles all Web interactions, including parsing dynamic HTML, and full support for all security requirements, such as user authentication or SSL protocol use.

A simple recorded script is ideal if your WebLOAD test involves a typical sequence of Web activities. These activities are all recorded in your script, and are represented in

the WebLOAD Recorder by a Script 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 script. Scripts 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 script to create more complex, sophisticated test sessions. For example, for a script to work with Java or COM components, a certain degree of programming is required. This guide documents the syntax of the JavaScript objects and functions available to programmers who wish to add more complex functionality to their scripts.

Scripts 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 scripts. WebLOAD JavaScript scripts 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 scripts, running test sessions, and analyzing the results, see the *WebLOAD Recorder User's Guide* and the *WebLOAD Console User's Guide*.

WebLOAD scripts Work with an Extended Version of the Standard DOM

WebLOAD Recorder operates in conjunction with a Web browser such as Microsoft's Internet Explorer. As you execute a sequence of HTTP actions in the browser, WebLOAD Recorder records your actions in a JavaScript script. 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 scripts use a standard browser DOM to access and navigate Internet Web pages, including Dynamic HTML and nested links and pages. To facilitate website 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 websites:

http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/introduction.html

http://msdn2.microsoft.com/en-us/library/ms533043.aspx

http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/dhtmlrefs.asp

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>
```

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/.

DOM Objects Commonly Used in a script

On Internet websites, 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.

DOM objects commonly used in scripts

The following table lists the DOM objects commonly used in scripts. A detailed description of each of these objects can be found in the following sections.

Table 3: DOM Objects Commonly Used in scripts

Object	Description
window	The `window` object represents an open browser 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.
document	The `document` object represents the HTML document in a browser window, storing the HTML data in a parsed format. Use the `document` object to retrieve links, forms, nested frames, images, scripts, and other information about the document. By default, `document` used alone represents the document in the current window. You usually refer directly to the `document;` the `window` part is optional and is understood implicitly.
frame	Each `frame` 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 `location` object contains information on the current window URL.
link	A `link` object contains information on an external document to which the current document is linked.

Object	Description
form, element, and input	A `form` object contains the set of elements and input controls (text, radio buttons, checkboxes, etc.) that are all components of a single form. Each element object stores the parsed data for a single HTML form element such as `<INPUT>`, `<BUTTON>`, or `<SELECT>`. Each input object stores the information defining one of the input controls in the form. Controls are organized by type, for example input type=checkbox. 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 `INPUT` or `SELECT`. Users viewing the form fill in the data and then click the `SUBMIT` 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.
image	Each `image` object contains one of the embedded images found in a document.
script	A `script` object defines a script for the current document that will be interpreted by a script engine.
title	The `title` object contains the document title, stored as a text string.

WebLOAD Extension Set

WebLOAD has added the following extensions to the standard DOM properties and methods. This guide provides syntax specifications for these objects.

WebLOAD DOM extension set highlights

Table 4: WebLOAD DOM Extension Set Highlights

WebLOAD object extensions	Description
wlCookie	Sets and deletes cookies.
wlException	WebLOAD error management object.
wlGeneratorGlobal and wlSystemGlobal objects	Handles global values shared between script threads or Load Generators.
wlGlobals	Manages global system and configuration values.
wlHeaders	Contains the key/value pairs in the HTTP command headers that brought the document. (Get, Post, etc.)
wlHttp	Performs HTTP transactions and stores configuration property values for individual transactions.

wlLocals	Stores local configuration property values.
wlMetas	Stores the parsed data for an HTML meta object.
wlOutputFile	Writes script output messages to a global output file.
wlRand	Generates random numbers.
wlSearchPairs	Contains the key/value pairs in a document's URL search strings.
wlTables, row, and cell objects	Contains the parsed data from an HTML table.
XML DOM objects	XML DOM object set that generates new XML data to send back to the server for processing.

Website 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 WebLOAD Recorder when initially creating scripts 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 this guide that are relevant to the HTTP Transaction Mode are noted as such in the entries.

When Would I Edit the JavaScript in My scripts?

WebLOAD Recorder automatically creates JavaScript scripts 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 scripts that were already recorded. Many users prefer to design test sessions around a set of basic scripts created through WebLOAD Recorder and then expand or tailor those scripts to meet a particular testing need. Some of the reasons for editing JavaScript scripts include:

- Recycling and updating a useful library of test scripts from earlier versions of WebLOAD.
- Creating advanced, specialized verification functions.
- Debugging the application being tested.

- Optimization capabilities, to maximize your application's functionality at minimal cost.

This guide documents the syntax and usage of the actions, functions, objects, and variables provided by WebLOAD to add advanced functionality and tailoring to the JavaScript scripts created through WebLOAD Recorder. 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.

Accessing script Components

WebLOAD uses test session scripts to simulate user activities at a website. A script is initially created by WebLOAD Recorder 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 the mouse), WebLOAD Recorder stores information about all user actions in a script. Scripts are also edited using WebLOAD Recorder. Users may add functionality or customize their scripts through the objects, functions, and other features described in this guide.

Customizing scripts may involve nothing more than dragging an icon from the WebLOAD Recorder toolbar and dropping it into a graphic representation of the script. 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 scripts by editing the underlying code of the script itself. When working with scripts, users may be working on many different levels. For that reason, the WebLOAD Recorder desktop includes multiple view options, providing information on multiple levels. See the *WebLOAD Scripting Guide* for a more extensive, illustrated explanation of the WebLOAD Recorder desktop components.

- Most users access scripts primarily through a *Script 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 Script Tree is referred to as a node. Nodes are organized in a hierarchical arrangement. The outmost level, or *root* level, is a single script node. The second level directly under the root script 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 Script Tree appears on the left side of the WebLOAD Recorder desktop.

- Web page nodes are added to the Script Tree in one of two ways. Some Web pages are the result of a user action on the previous page, such as clicking 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 a Script Tree parallel the *navigation blocks* found within the JavaScript script code.

  During a WebLOAD Recorder recording session, a new navigation block is created each time a user completes a direct navigation, manually entering a new URL into the WebLOAD Recorder address bar. Each navigation block is surrounded by a `try{} catch{}` statement in the corresponding JavaScript script 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 Script Tree.

- The graphic nodes in a Script Tree actually represent blocks of code within the underlying recorded script. 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 WebLOAD Recorder desktop.
- The graphic nodes in a Script Tree represent user actions on a website. An exact replica, or snapshot, of each user activity is stored during recording and available in the 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 WebLOAD Recorder 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 WebLOAD Recorder 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 WebLOAD Recorder 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 WebLOAD Recorder desktop displaying the Script Tree and DOM and Browser Views. The Script Tree is on the left. 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 script 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.

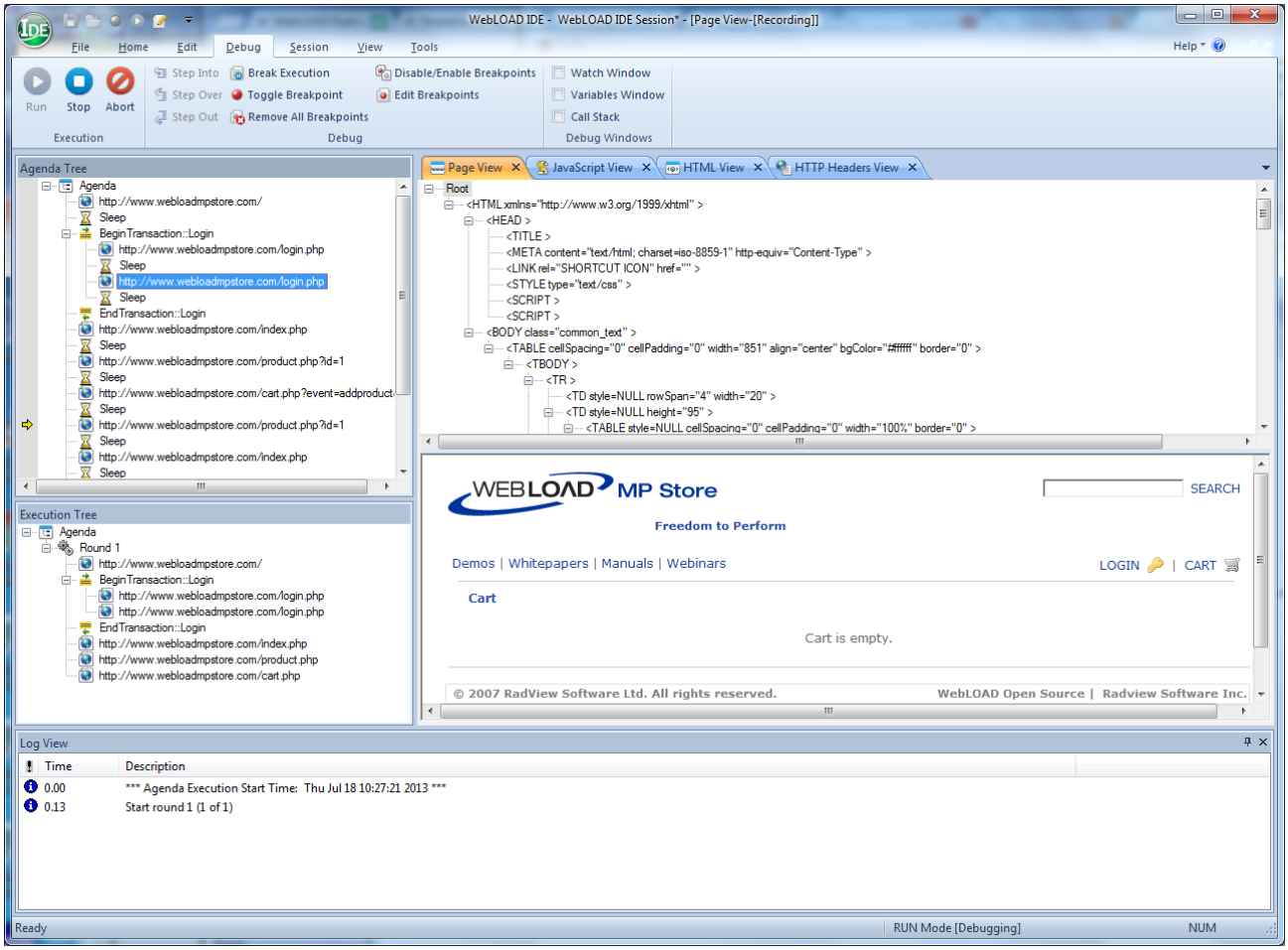

Figure 1: Script Tree, DOM, and Browser Views

Editing the JavaScript Code in a script

Accessing the JavaScript Code within the Script Tree

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

While most people never really work with the JavaScript code within their script, some users do wish to manually edit the JavaScript code underlying their Script Tree. For example, some test sessions may involve advanced WebLOAD testing features that cannot be completely implemented though the GUI, such as Java or XML objects. Editing the JavaScript code in a script 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. WebLOAD Recorder provides access to the JavaScript code in a script through JavaScript Object nodes, which are seen on the following levels:

- JavaScript Object nodes—individual nodes in the Script Tree. Empty JavaScript Object nodes may be dragged from the WebLOAD Recorder toolbar and dropped onto the Script Tree at any point selected by the user, as described in the *WebLOAD Scripting Guide*. Use the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18), to add lines of code or functions to the JavaScript Object.
- Converted Web page—the sub-tree or branch of a Script 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 Script 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 script, 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 the preferred Web page node in the Script 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: 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 script. Select **Edit ➤ Import JavaScript File** from the

WebLOAD Recorder 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 Script Tree—if necessary, an entire Script Tree can be converted to a single JavaScript Object node consisting of a straight JavaScript text file. Right click the Script 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 script file is truly required for the test session.

Using the IntelliSense JavaScript Editor

For those users who wish to manually edit their scripts, WebLOAD Recorder provides three levels of programming assistance:

- An IntelliSense Editor mode for the JavaScript View pane.

  Add new lines of code to your script or edit existing JavaScript functions through the IntelliSense Editor mode of the JavaScript View pane. The IntelliSense Editor helps you 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. IntelliSense supports the following shortcut keys:

  - **Period (".")** – Enter a period after the object name, to display a drop-down list of the object's available properties that can be added to the script (see Figure 2).
  - **<CTRL> <Space>** – While typing the name of an object, you can type <CTRL> <Space> to display a drop-down list of the available objects that begin with the letters that you entered. For example, if you type `wl` the IntelliSense Editor displays a drop-down list of all of the objects that begin with `wl` (such as `wlhttp`).

  In addition, the IntelliSense Editor gives a structure to the code with the outline bar and line numbering.

  Collapsing the code enables you to view the heading of the section, without seeing the code within the section. To expand or collapse different sections of the code:

  - Click the plus sign (+) or minus sign (-) on the outline bar,

  -Or-

  - Right-click within the IntelliSense Editor and select Outlining from the pop-up menu. The available outlining options are:

- **Toggle outline** – Collapses or expands the section at the mouse location.
- **Toggle all outline** – Collapses or expands all outlines.
- **Collapse to definition** – Collapses all outlines.

You can enable or disable both the outline bar and line numbering features by:

- Selecting **Edit ➤ Enable Outlining** or **Line Numbers**,

-Or-

- Right-clicking within the IntelliSense Editor and selecting **Enable Outlining** or **Line Numbers** from the pop-up menu.

When these features are enabled, a checkmark appears next to the name in the Edit and pop-up menus. By default, these features are enabled, but WebLOAD opens with the settings that were saved during the previous WebLOAD session. During playback and debug modes, all outlines are expanded.

Use WebLOAD Recorder's predefined delimiters to keep your code structured and organized. The available delimiters include:

- For JavaScript functions, use the "{" as the start delimiter and the "}" end delimiter.
- For script tree nodes, insert a WLIDE comment from the General WebLOAD Recorder toolbox. This automatically inserts a start delimiter "//" and end delimiter "End WLIDE".

For more information, see the *WebLOAD Scripting Guide.*

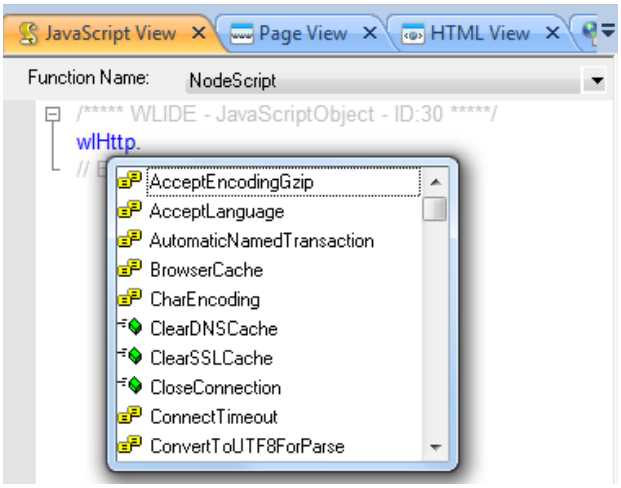

Figure 2: IntelliSense Editor Mode for JavaScript View Pane

- A selection of the most commonly used functions and commands, available through the **Insert** menu.

You can choose to program your own JavaScript Object code within your script and take advantage of the WebLOAD Recorder to simplify your programming efforts. Rather than manually typing out the code for each command, with the risk of making a mistake, even a trivial typographical error, and adding invalid code to the script file, you may select an item from the **Insert** menu, illustrated in the following figure, to bring up a list of available commands and functions for the selected item. WebLOAD Recorder automatically inserts the correct code for the selected item into the JavaScript Object currently being edited. You may then change specific parameter values without any worries about accidental mistakes in the function syntax.

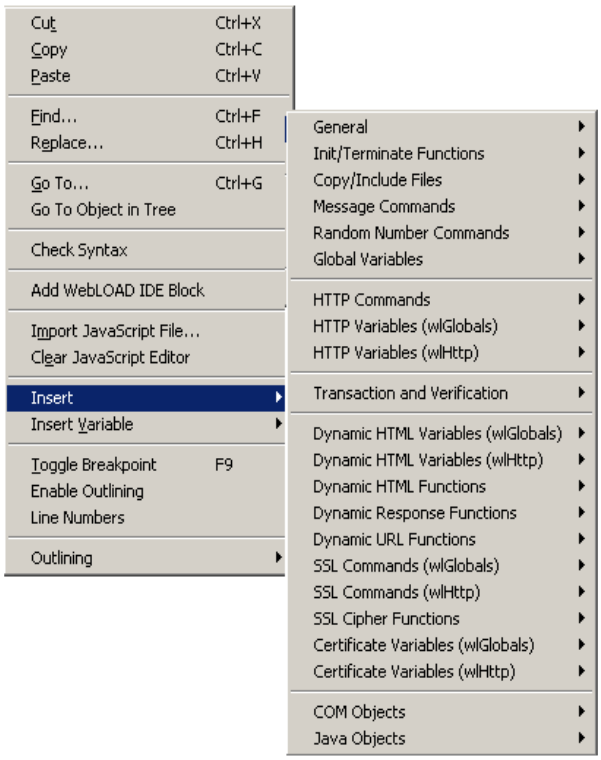

Figure 3: Insert Menu

In addition to the Insert menu, you may select an item from the Insert Variable menu, to add system and user-defined parameters to the script. This eliminates the need for manual coding.

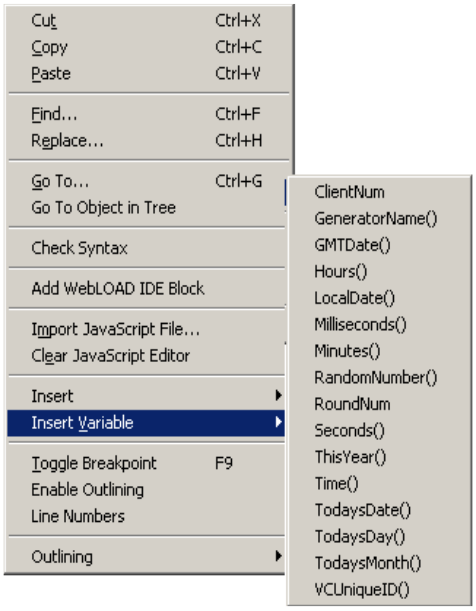

Figure 4: Insert Variable Menu

- A Syntax Checker that checks the syntax of the code in your script file and catches simple syntax errors before you spend any time running a test session. While standing in the JavaScript View pane of the WebLOAD Recorder desktop, select **Tools ➤ Check Syntax** to check the syntax of the code in your script file.

Important: WebLOAD Recorder scripts should be edited only within the confines of WebLOAD Recorder, not an external editor. If you use an external editor to modify the JavaScript code in a script file generated by WebLOAD Recorder, your visual script will be lost.

Script code that you wish to write or edit must be part of a JavaScript Object in the Script Tree. Adding or converting JavaScript Objects in a Script Tree is described in *Accessing the JavaScript Code within the Script Tree* (on page 17).

Chapter 3

Using the WebLOAD JavaScript Reference

The WebLOAD JavaScript programming tools provide a powerful means of adding sophisticated, complex tailoring to recorded scripts. WebLOAD supports literally hundreds of functions, objects, properties, and methods, to provide optimal programming power for your test session script.

To simplify access to the WebLOAD JavaScript toolset, this section organizes the functions and objects into major categories, providing you with information to help you locate a specific tool or capability.

Note: These categories do not constitute an exhaustive list of all WebLOAD JavaScript objects, properties, methods, and functions. This is simply a list of the major categories, to help you quickly identify the most commonly used items.

The WebLOAD JavaScript toolset includes many additional elements. For a complete, alphabetical reference list of all toolset components, see *WebLOAD Actions, Objects, and Functions* (on page 37).

The WebLOAD JavaScript toolset can be organized into the following categories:

- **Collections**—Meta-objects that serve as arrays or sets of individual objects. Described in *Collections* (on page 27).
- **File Management**—Functions used to manage access to a script's external files. Described in *File Management Functions* (on page 28).
- **Identification Components**—Functions and variables used to identify specific elements or points of time during a test session, for clarity in understanding session results and output reports. Described in *Identification Variables and Functions* (on page 29).
- **Message Functions**—Functions used to display messages in the WebLOAD Console Log Window. Described in *Message Functions* (on page 30).
- **Objects**—A brief introduction to the WebLOAD JavaScript object set. See *Objects* (on page 32).

- **SSL Cipher Command Suite**—A set of functions and properties that implement full SSL/TLS 1.0 protocol support. Described in *SSL Cipher Command Suite* (on page 33).
- **Timing Functions**—Functions used to time or synchronize any operation or group of user activities in a script. Described in *Timing Functions* (on page 34).
- **Transaction Verification Components**—Components used to create customized transaction verification functions. Described in *Transaction Verification Components* (on page 36).
- **Parameterization**—A brief introduction explanation and reference to all parameterization objects and functions. Described in *Parameterization* (on page 35).
- **Internet Protocol Support**—Objects that implement full support of the complete range of Internet protocols. Described in *WebLOAD Internet Protocols* Reference (on page 347).

HTTP Components

Properties and Methods of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The `wlGlobals`, `wlLocals`, and `wlHttp` objects share a set of components that manage user HTTP 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.

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.

Syntax

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

Example

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

Methods

- ClearDNSCache() (see *ClearDNSCache() (method)* on page 49)
- ClearSSLCache() (see *ClearSSLCache() (method)* on page 49)

The following methods are for wlHttp objects only:

- CloseConnection() (see *CloseConnection() (method)* on page 53)
- Get() (see *Get() (transaction method)* on page 104)
- Post() (see *Post() (method)* on page 205)
- Head() (see *Head() (method)* on page 139)

Data Methods

- wlClear() (see *wlClear() (method)* on page 301)
- wlGet() (see *wlGet() (method)* on page 310)
- wlSet() (see *wlSet() (method)* on page 328)

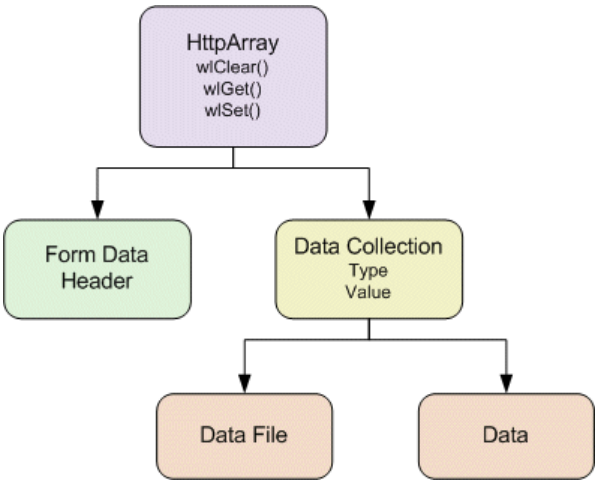

Figure 5: wlHttp Array

Properties

The following properties are for wlHttp objects only

Data Properties

- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Header (see *Header (property)* on page 140)

- DataCollection.type (see *type (property)* on page 288)
- DataCollection.value (see *value (property)* on page 294)

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

Configuration Properties

- ConnectionSpeed (see *ConnectionSpeed (property)* on page 55) `(wlGlobals only)`
- DisableSleep (see *DisableSleep (property)* on page 76)
- DNSUseCache (see *DNSUseCache (property)* on page 77)
- KeepAlive (see *KeepAlive (property)* on page 159)
- LoadGeneratorThreads (see *LoadGeneratorThreads (property)* on page 165)
- MultiIPSupport (see *MultiIPSupport (property)* on page 171)
- NTUserName, NTPassWord (see *NTUserName, NTPassWord (properties)* on page 176)
- Outfile (see *Outfile (property)* on page 188)
- PassWord (see *PassWord (property)* on page 203)
- ProbingClientThreads (see *ProbingClientThreads (property)* on page 208)
- Proxy, ProxyUserName, ProxyPassWord (see *Proxy, ProxyUserName, ProxyPassWord (properties)* on page 210)
- RedirectionLimit (see *RedirectionLimit (property)* on page 216)
- SaveSource (see *SaveSource (property)* on page 226)
- SaveTransaction (see *SaveTransaction (property)* on page 226) `(wlGlobals only)`
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) `(wlGlobals only)`
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) `(wlGlobals only)`
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- type (see *type (property)* on page 288)
- Url (see *Url (property)* on page 289)
- UserAgent (see *UserAgent (property)* on page 291)
- UserName (see *UserName (property)* on page 291)
- UsingTimer (see *UsingTimer (property)* on page 292)

- Version (see *Version (property)* on page 299)
- wlTarget (see *wlTarget (property)* on page 334)

See also

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

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 scripts work with all browser DOM collections and objects. The recommended way to access these objects is through the classic browser `document` object, via the relevant collection. For example, access a table through:

```
document.links[0]
```

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 `images` objects are contained within the `images` collection of a document, check the value of:

```
document.images.length
```

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

See also

- element (see *element (object)* on page 80)

File Management Functions

Description

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

Note: Input file management is also provided by wlInputFile (see *wlInputFile (object)* on page 317). Output file management is also provided by wlOutputFile (see *wlOutputFile (object)* on page 323).

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- GetLine() (`wlOutputFile`) (see *GetLine() (function)* on page 123)
- GetLine() (`wlInputFile`) (see *GetLine() (method)* on page 125)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (`wlOutputFile`) (see *Open() (function)* on page 183)
- Open() (`wlInputFile`) (see *Open() (method)* on page 180)
- Reset() (see *Reset() (method)* on page 220)
- Using the IntelliSense JavaScript Editor (see *Using the IntelliSense JavaScript Editor* on page 18)
- wlOutputFile() (see *wlOutputFile (object)* on page 323)
- wlInputFile() (see *wlInputFile (object)* on page 317)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

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 (see *ClientNum (property)* on page 50) and RoundNum, (see *RoundNum (variable)* on page 222) identify the client and round number of the current script instance.
- The GeneratorName() (see *GeneratorName() (function)* on page 101) function identifies the current Load Generator.
- The GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131) function identifies the operating system of the current Load Generator.
- The VCUniqueID() (see *VCUniqueID() (function)* on page 296) 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 script 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 WebLOAD Recorder. All the variables that appear in this list are available for use at all times in a script file. In the WebLOAD Recorder main window, click **Variable Windows** in the **Debug** tab of the ribbon..

For example, it is convenient to add `ClientNum` to a Message Node to clarify which client sent the messages that appear in the WebLOAD Console Log window.

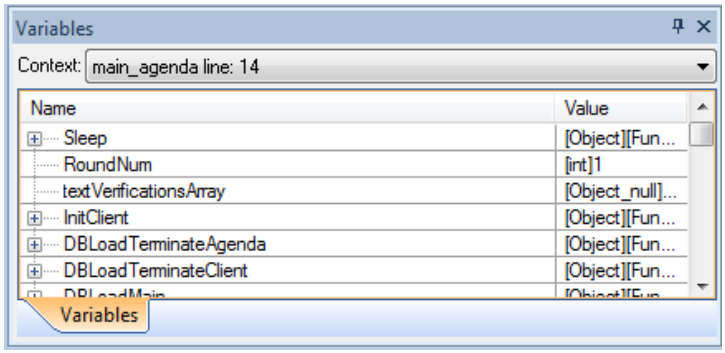

Figure 6: Variables List in WebLOAD Recorder

See also

- ClientNum (see *ClientNum (property)* on page 50)
- GeneratorName() (see *GeneratorName() (function)* on page 101)
- GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131)
- RoundNum (see *RoundNum (variable)* on page 222)
- VCUniqueID() (see *VCUniqueID() (function)* on page 296)

Message Functions

Description

These functions display messages in the Log Window of WebLOAD Recorder 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 Console User's Guide*.

Example

In the following example, the script 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: Message functions are usually accessed and inserted into script files directly through the WebLOAD Recorder. Message function commands can be added to the script in Visual Editing mode using the Toolbox message item and the Insert menu command. The JavaScript code line that corresponds to this message function appears in the JavaScript View pane.
Message function command lines may also be added directly to the code in a JavaScript Object within a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).
Messages can also be added to the script using the Toolbox Message icon  . Drag the Message icon to the Script Tree. The Message dialog box appears. Type or select the information to appear in the message. Use double quotes to include a string value, or click  to select a variable. Select the severity of the message from the Message Severity drop-down list.

See also

- *Error Management* in the *WebLOAD Scripting Guide*
- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException (see *wlException (object)* on page 306)
- wlException() (see *wlException() (constructor)* on page 308)

Objects

Description

WebLOAD scripts Work with an Extended Version of the Standard DOM on page 6 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 websites:

- 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 HTTP 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 scripts are listed here. This guide 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.

Website testing usually means testing how typical user activities are handled by the application being tested. Are the user actions managed quickly, correctly and 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 WebLOAD Recorder when initially creating scripts 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 Guide* that are relevant to the HTTP Transaction Mode are noted as such in the entry.

SSL Cipher Command Suite

Description

WebLOAD provides full SSL/TLS 1.0/TLS 1.2 protocol support through 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 *SSL Ciphers – Complete List* on page 450.

Functions

The Cipher Command Suite includes the following functions:

- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page *261*)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLEnableStrength() (see *SSLEnableStrength() (function)* on page262)

Comment

Use the Cipher Command Suite to check or verify SSL configuration information at any point in your script. However, any *changes* to a script's SSL property configuration, whether through the `wlGlobals` properties or the Cipher Command Suite functions, must be made in the script's *initialization functions*. Configuration changes made in the `InitAgenda()` function will affect all client threads spawned during that script'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 a script's main body. The results will be undefined for all subsequent transactions.

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (wlGlobals only)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)

- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (wlGlobals only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page *261*)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLEnableStrength() (see *SSLEnableStrength() (function)* on page 262)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Timing Functions

Description

The timer functions let you time or synchronize any operation or group of user activities in a script, such as a navigation or mouse click, and send the time statistics to the WebLOAD Console.

Example

The following script connects to the home Web page of company. On every fifth round, the script also connects to a second Web page. The script uses different timers to measure the time for each connection.

Note: This script 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() (see *SendCounter() (function)* on page 237)
- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)
- Sleep() (see *Sleep() (function)* on page 248)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)

Parameterization

Parameterization enables you to edit a script containing static values and transform it into a script that will run multiple variations of the static values.

When recording a script, WebLOAD captures the data that is being sent, including login details, user selections, and entered text. When running the script under load, simulating many users, it is desirable to use variations in the data, so as to simulate the application more realistically. To do so, you can replace the static values with parameters.

Parameter values can come from a file, or be automatically generated numbers, strings and dates.

Using parameterization enables you to specify how the script should select values from the data file. For example:

- Order considerations – Whether to randomly select values from the data file, or use them in the order they appear.
- Uniqueness considerations – Whether the same value can be used at the same time by different virtual clients.

You can also specify the update policy, which defines when a new value will be read or calculated. For example, whether to update the value on each round, or once at the beginning of the test.

In addition to defining a data file, you can also use parameterization to define a random number format, date/time format, and string format. These can also be used to replace static values. For example, if the online shop delivers books between 1-14 days from the date of purchase, you can define a random number format of 1-14 and replace the static desired delivery period value with a call to the random number format.

Functions

The set of parameterization functions includes the following:

- wlTimeParam() (see *wlTimeParam() (parameterization)* on page 335)
- wlDataFileParam() (see *wlDataFileParam() (parameterization)* on page 304)
- wlNumberParam() (see *wlNumberParam() (parameterization)* on page 321)
- wlStringParam() (see *wlStringParam() (parameterization)* on page 331)

Transaction Verification Components

Description

Customized transaction verification functions are created out of the following components:

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

See also

- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)

Chapter 4

WebLOAD Actions, Objects, and Functions

This chapter includes syntax specifications for the objects, properties, methods, and functions most useful for users who wish to program the code within their JavaScript scripts. To simplify and clarify the information presented, this chapter begins with a brief introduction to the concept of the basic Document Object Model, or DOM, upon which most website implementations are based. After this basic introduction, the rest of the chapter consists of reference entries for each item, arranged in alphabetical order.

AcceptEncodingGzip (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Sets the wlGlobals.AcceptEncodingGzip flag, which enables Gzip support. For each request, WebLOAD sends the header "Accept-Encoding: gzip, deflate". This tells the server that the client can accept zipped content. As most servers will work correctly even if the client does not send the "Accept-Encoding: gzip, deflate" header, it is recommended not to set the wlGlobals.AcceptEncodingGzip flag because it is performance heavy. However, some servers will fail if it is not sent. The default value of AcceptEncodingGzip is **false**.

You may want to test your application in GZIP mode in the following cases:

- The server only works in GZIP mode and rejects any requests that do not enable GZIP mode.
- GZIP is enabled and the server supports non-GZIP requests. A non-GZIP request means that the web server does less work, but places more stress on the network for large responses. This is acceptable if you are testing a back end server.

However, if you realistically want to test an end-to-end system, enable GZIP support.

GUI mode

In WebLOAD Recorder, select or deselect the **GZip Support** checkbox in the Browser Parameters tab of the **Default** or **Current Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Console, select or deselect the **GZip Support** checkbox in the Browser Parameters tab of the **Default** or **Current Options** dialog box or the **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

Example

```
a.AcceptEncodingGZip = true
```

See also

- *HTTP Components* (on page 24)

AcceptLanguage (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Sets the wlGlobals.AcceptLanguage flag, which defines a global value for the AcceptLanguage header. Some applications/servers will behave differently depending on this setting. The AcceptLanguage flag is a simple string and WebLOAD does not enforce any checks on the values assigned to it.

Example

```
wlGlobals.AcceptLanguage = "En-us"
```

GUI mode

In WebLOAD Recorder, select or deselect the **Accept Language** checkbox in the HTTP Parameters tab of the **Default** or **Current Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Console, select or deselect the **Accept Language** checkbox in the HTTP Parameters tab of the **Default** or **Current Options** dialog box or the **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

Some Asian sites check the AcceptLanguage property, and, if they think the client is working in English, the flow might not be exactly as recorded.

action (property)

Property of Object

- form (see *form (object)* on page 95)

Description

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

Example

```
Document.forms[0].action
```

Add() (method)

Method of Objects

- wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309)
- wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)

Description

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

Syntax

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

Parameters

Parameter Name	Description
SharedIntVarName	The name of a shared integer variable to be incremented.
number	An integer with the amount to add to the specified variable.

Parameter Name	Description
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 script, 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 scripts, 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 script, 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 scripts, run by all processes, on all Load Generators, system-wide.

Example

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

See also

- Get() (see *Get() (addition method)* on page 102)
- Set() (see *Set() (addition method)* on page 240)

AuthType (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Specifies the authentication method to be used by the server: Kerberos or NTLM. The default value is **NTLM**.

Note: The AuthType property is only relevant for playback.

Example

```
wlGlobals.AuthType = "Kerberos"
```

GUI mode

To set the authentication method to be used by the server:

- In WebLOAD Console, select the authentication method in the Authentication tab of the **Default, Current Session, or Script Options** dialog box, accessed from the **Tools** tab of the ribbon.
- In WebLOAD Recorder, select the authentication method in the Authentication tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

If the `AuthType` property was set to "Kerberos" and the server does not support Kerberos, WebLOAD will automatically change the authentication method to "NTLM".

See also

- KDCServer() (see *KDCServer (property)* on page 158)

Async (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)
- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Causes the HTTP request to be asynchronous.

The possible values of `wlHttp.Async` are:

- **false** – the following HTTP command is synchronous (default).
- **true** – the following HTTP command is asynchronous.

  When using asynchronous requests, the script does not wait for the request to complete before moving on to the next statement. In order to work with the response, you need to specify one of the asynchronous callback functions – *onDocumentComplete (property)* and/or *onDataReceived (property)*.

Example

```
wlHttp.Async = true;
wlHttp.onDocumentComplete = function(document) {
   InfoMessage("Response " + document.wlSource);
}
wlHttp.Get("http://something");
```

See also

- *HTTP Components* (on page 24)
- *onDocumentComplete (property)* (on page 179)
- *onDataReceived (property)* (on page 177)
- The *Using Asynchronous Requests* chapter in the *WebLOAD Scripting Guide*

BeginTransaction() (function)

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.

Optionally, you can specify a period of time, which is the minimum amount of time for the transaction. If the total time of the transaction is less than the time period specified, the machine sleeps for the remainder of the time in order to simulate the intermittent activity of real users.

The behavior of the sleep time is affected by the Sleep Time Control settings that are set in the Current Project Options of the WebLOAD Recorder and Console. These settings can be one of the following:

- **Sleep time as recorded** – Runs the script with the delays corresponding to the natural pauses that occurred when recording the script.
- **Ignore recorded sleep time (default)** – Eliminates any pauses when running the script and runs a worst-case stress test.
- **Set random sleep time** – Sets the ranges of delays to represent a range of users.
- **Set sleep time deviation** – Sets the percentage of deviation from the recorded value to represent a range of users.

For more information on setting the Sleep Time Control settings, see *Configuring Sleep Time Control Options* in the *WebLoad Recorder User's Guide*.

Note: If the transaction fails, it still sleeps for the specified time interval. This is true even if an error not directly connected to the transaction is received, for example, HTTP 500 for a GET within the transaction.

Syntax

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

Parameters

Parameter Name	Description
`TransName`	The name assigned to this transaction, a user-supplied string.
`SleepTime`	An integer value specifying the interval of time (in milliseconds) for the minimum amount of time for the transaction. .

GUI mode

Note: `BeginTransaction()` and `EndTransaction()` functions are usually accessed and inserted into script files directly through the WebLOAD Recorder. For example, the following figure illustrates a section in the Script Tree bracketed by BeginTransaction and EndTransaction nodes. The EndTransaction node is highlighted in the Script Tree.

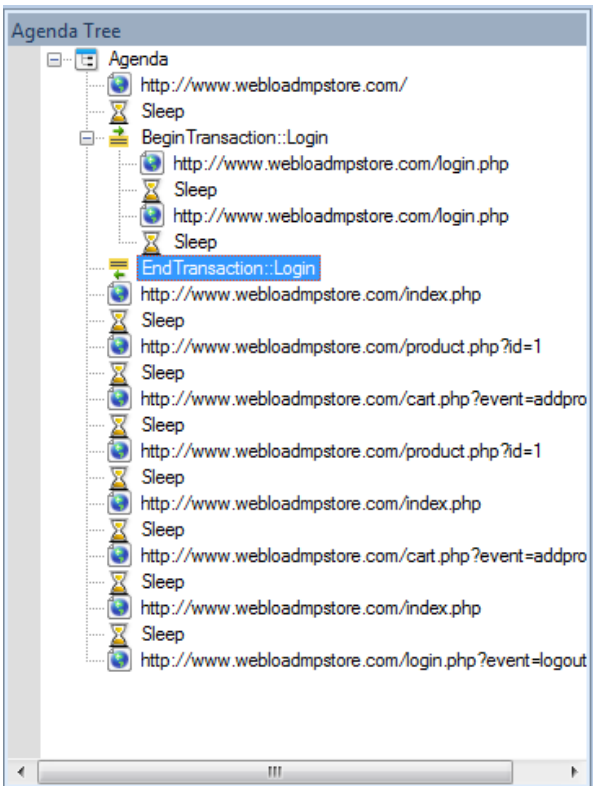

Figure 7: Form Branch in Script Tree Bracketed by BeginTransaction and EndTransaction Nodes

To mark the beginning of a transaction, drag the **Begin Transaction** icon from the Load toolbox into the Script Tree, directly above the first action you want to include in the transaction. The Begin Transaction dialog box opens. For additional information about the `BeginTransaction()` function, refer to *Begin and End Transaction* in the *WebLOAD Recorder User's Guide*.

See also

- EndTransaction() (see *EndTransaction() (function)* on page 88)

- CreateTable() (see *CreateTable() (function)* on page 65)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- *Transaction Verification Components* (on page 36)
- TransactionTime (see *TransactionTime (property)* on page 287)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

cell (object)

Property of Objects

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

- row (see *row (object)* on page 223)
- wlTables (see *wlTables (object)* on page 333)

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 `wlTables` 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: This is not typical-the standard JavaScript index begins at **0** to represent the first element in a set.

Example

For cells within a wlTables object:

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

Note: 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 (see *cellIndex (property)* on page 46)
- InnerHTML (see *InnerHTML (property)* on page 154)
- InnerText (see *InnerText (property)* on page 156)
- tagName (see *tagName (property)* on page 279)

Comment

`cell` is often accessed through the wlTables family of table, row, and cell objects.

See also

- cellIndex (see *cellIndex (property)* on page 46) `(cell` property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) `(wlTables` property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)

- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (wlTables property)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell property)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (wlTables property)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- tagName (see *tagName (property)* on page 279) (cell property)
- wlTables (see *wlTables (object)* on page 333)

cellIndex (property)

Property of Object

- cell (see *cell (object)* on page 44)

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`.

Comment

`cellIndex` is a member of the wlTables family of table, row, and cell objects.

See also

- cell (see *cell (object)* on page 44) (`wlTables` and `row` property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (`wlTables` property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (`wlTables` property)
- InnerHTML (see *InnerHTML (property)* on page 154) (`cell` property)

- InnerText (see *InnerText (property)* on page 156) `(cell` property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) `(wlTables` property)
- rowIndex (see *rowIndex (property)* on page 224) `(row` property)
- tagName (see *tagName (property)* on page 279) `(cell` property)
- wlTables (see *wlTables (object)* on page 333)

CharEncoding (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Contains the value corresponding to the character set being used. The default value is **Default** (0), the regional settings of the computer. For a complete list of the supported character sets, see *WebLOAD–supported Character Sets* on page 479.

Example

If you want to specify that you are using Japanese (EUC), set the value of CharEncoding as follows:

```
wlGlobals.CharEncoding = 51932
```

GUI Mode

In WebLOAD Console, select a character set in Character Encoding list box in the Browser Parameters tab of the **Default Options** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, select a character set in the Character Encoding list box in the Browser Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- EnforceCharEncoding (see *EnforceCharEncoding (property)* on page 87)

checked (property)

Property of Object

- element (see *element (object)* on page 80)

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).

ClearAll() (method)

Method of Object

- wlCookie (see *wlCookie (object)* on page 302)

Description

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

Syntax

```
wlCookie.ClearAll()
```

ClearCookiesAtEndOfRound (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Indicates whether to clear the cookies at the end of each round. The default value of `ClearCookiesAtEndOfRound` is **true**. By setting this flag to false, the cookies list will not be cleared at the end of each round.

Example

```
wlGlobals.ClearCookiesAtEndOfRound = false
```

ClearDNSCache() (method)

Method of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Clear the IP address cache.

Syntax

```
wlHttp.ClearDNSCache()
```

GUI mode

In WebLOAD Console, disable caching for the Load Generator or for the Probing Client during a test session by clearing the appropriate box in the Browser Parameters tab of the **Default**, **Current Session Options** or **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, disable caching during execution by clearing the appropriate box in the Browser Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

To enable or disable DNS caching, set the DNSUseCache (see *DNSUseCache (property)* on page 77) property.

See also

- *HTTP Components* (on page 24)
- ClearSSLCache() (see *ClearSSLCache() (method)* on page 49)
- DNSUseCache (see *DNSUseCache (property)* on page 77)
- SSLUseCache (see *SSLUseCache (property)* on page 272)

ClearSSLCache() (method)

Method of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Clear the SSL decoding-key cache.

Syntax

```
wlHttp.ClearSSLCache()
```

GUI mode

In WebLOAD Console, disable the SSL cache for the Load Generator or for the Probing Client during a test session by clearing the appropriate box in the Browser Parameters tab of the **Default**, **Current Session Options** or **Script Options** dialog box, accessed from the **Tools** tab of the ribbon..

In WebLOAD Recorder, disable the SSL cache during execution by clearing the appropriate box in the Browser Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon..

Comment

To enable or disable SSL caching, set the SSLUseCache (see *SSLUseCache (property)* on page 272) property.

See also

- *HTTP Components* (on page 24)
- ClearDNSCache() (see *ClearDNSCache() (method)* on page 49)
- DNSUseCache (see *DNSUseCache (property)* on page 77)
- SSLUseCache (see *SSLUseCache (property)* on page 272)

ClientNum (property)

Description

`ClientNum` is set to the serial number of the client in the WebLOAD test configuration. `ClientNum` is a read-only local property. Each client in a Load Generator has a unique `ClientNum`. However, two clients in two different Load Generators may have the same `ClientNum`.

Note: While `ClientNum` is unique within a single Load Generator, it is not unique system wide. Use VCUniqueID() (see *VCUniqueID() (function)* on page 296) 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, ..., N-1`. You can access `ClientNum` anywhere in the local context of the Script (`InitClient()`, main script, `TerminateClient()`, etc.). `ClientNum` does not exist in the global context (`InitAgenda()`, `TerminateAgenda()`, etc.).

If you mix Scripts within a single Load Generator, instances of two or more Scripts 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 (property)* on page 165).

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 property, through the WebLOAD Recorder. The variables that appear in this list are available for use at any point in a script file. In the WebLOAD Recorder main window, click **Variable Windows** in the **Debug** tab of the ribbon..

For example, it is convenient to add `ClientNum` to a Message Node to clarify which client sent the messages that appear in the WebLOAD Console Log window.

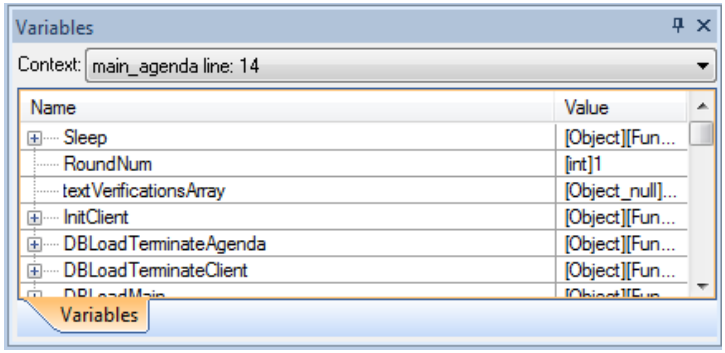

Figure 8: Variables Window

See also

- GeneratorName() (see *GeneratorName() (function)* on page 101)
- GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131)
- *Identification Variables and Functions* (on page 29)
- RoundNum (see *RoundNum (variable)* on page 222)
- ThreadNum (see *ThreadNum() (property)* on page 282)
- VCUniqueID() (see *VCUniqueID() (function)* on page 296)

Close() (function)

Method of Object

- wlOutputFile (see *wlOutputFile (object)* on page 323)

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

Parameter Name	Description
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.

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

- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)

- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile (see *wlOutputFile (object)* on page 323)
- wlOutputFile() (see *wlOutputFile (object)* on page 323)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

CloseConnection() (method)

Method of Object

- wlHttp (see *wlHttp (object)* on page 316)

Description

Closes all open connections. If `CloseConnection()` is not called, all connections that were opened with the KeepAlive option (see *KeepAlive (property)* on page 159) remain open until the end of the round. HTTP connections are automatically closed at the end of each round.

Syntax

```
wlHttp.CloseConnection()
```

GUI mode

WebLOAD recommends maintaining or closing connections through the WebLOAD Console. Enable maintaining connections for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default Options** dialog box, accessed from the **Tools** tab of the ribbon..

In WebLOAD Console, enable maintaining connections for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default Options** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon..

In WebLOAD Recorder, enable maintaining connections during execution by checking the appropriate box in the Browser Parameters tab of the **Tools ➤ Default** or **Current Project Options** dialog box.

See also

- *HTTP Components* (on page 24)

- KeepAlive (see *KeepAlive (property)* on page 159)

cols (property)

Property of Object

- element (see *element (object)* on page 80)
- wlTables (see *wlTables (object)* on page 333)

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.

Comment

`cols` is often accessed through the `wlTables` family of table, row, and cell objects.

See also

- cell (see *cell (object)* on page 44) `(wlTables` and `row` property)
- cellIndex (see *cellIndex (property)* on page 46) `(cell` property)
- *Collections* (on page 27)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) `(wlTables` property)
- InnerHTML (see *InnerHTML (property)* on page 154) `(cell` property)
- InnerText (see *InnerText (property)* on page 156) `(cell` property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) `(wlTables` property)
- rowIndex (see *rowIndex (property)* on page 224) `(row` property)
- tagName (see *tagName (property)* on page 279) `(cell` property)

ConnectTimeout (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

ConnectTimeout is used to set the amount of time the system will wait for an HTTP connection to be established before timing out. The ConnectTimeout property is defined in milliseconds. Use the ConnectTimeout property to fine tune the Load Generator performance.

Example

```
wlGlobals.ConnectTimeout = 7
```

See also

- *HTTP Components* (on page 24)

ConnectionSpeed (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

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 website 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 14,400 bps, 28,800 bps, etc.

Note: The typical single ISDN line can carry 64 Kb, a double line carries 128 Kb, and a T1 line can handle 1.5 Mb.

Syntax

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

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

GUI mode

WebLOAD recommends setting the connection speed through the WebLOAD Console. 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 **Default Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)

content (property)

Property of Object

- wlMetas (see *wlMetas (object)* on page 320)

Description

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

Syntax

`wlMetas[index#].content`

Example

```
document.wlMetas[0].content
```

See also

- httpEquiv (see *httpEquiv (property)* on page 144)
- Name (see *Name (property)* on page 174)
- Url (see *Url (property)* on page 289)

ContentLength (function)

Description

Verifies the content length of the service response.

Syntax

```
wlVerification.ContentLength(operator, length, severity)
```

Parameters

Parameter Name	Description
operator	One of the following mathematical operators: • < - less than. • > - greater than. • = - equal to.
length	The expected length of the content in bytes.
severity	Possible values of this parameter are: • `WLSuccess`. The transaction terminated successfully. • `WLMinorError`. This specific transaction failed, but the test session may continue as usual. The Script 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 Script 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 Script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

The following code verifies that the page content length is equal to 120 bytes. In case of failure, WebLOAD displays a fatal error and stops the execution.

```
wlVerification.ContentLength("=" , 120, WLSevereError);
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- Severity (see *Severity (property)* on page 247)

- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

ContentType (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Specifies the content type of the HTTP request.

Example

```
wlGlobals.ContentType = "text/html"
```

See also

- *HTTP Components* (on page 24)

ConvertHiddenFields(method)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Converts hidden fields to dynamic values. This is done by correlate the Script so it uses the dynamic value of the field, not the value recorded in the Script.

The ConvertHiddenFields method takes the URL to be submitted via a Get or Post action and searches for it in the current DOM. This is done by looping over the document.form[] collection until it finds a form whose action matches the URL. It then loops over its elements[] collection. Each element whose type is "hidden" is then inserted into the wlHttp.FormData collection, overriding any existing value. The recorded values are replaced by the dynamic values during playback.

Note: ConvertHiddenFields cannot be accessed directly by the user. See the example in the Comment section below.

Syntax

Use

```
SaveCurrentHiddenFields(url)
```

after the page with the fields and specify the URL of the page.

Comment

Because the WebLOAD Recorder does not filter internal frames of a page, there are cases when the data required for the correlation will not be found in the DOM of the previous request.

For example:

The page you are working with is called frame1.html and is an internal frame of a page called page.html, which has four internal frames (frame1 - frame 4). You recorded a navigation to page.html and then submitted the form on frame1.html. Thus, your Script would appear as follows:

```
Get page.html
Get frame1.html
Get frame2.html
Get frame3.html
Get frame4.html
Post the form from frame1.html
```

In order to correlate the data for the final Post, you need the document from frame1. The intervening Get's, however, will not enable you to get this document. Manually insert the SaveCurrentHiddenFields() method after frame1.html in this example. This method saves the hidden fields so that the automatic correlation can use it when needed.

CookieDomain (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

When set to `true`, the client checks if the cookie domain matches the request domain during GET/POST. Use this property if you need to emulate the setting of client side cookies or modify server cookies on the client side.

Note: This property can only be inserted manually.

Example

```
wlGlobals.CookieDomain = false
```

See also

- CookieExpiration (see *CookieExpiration (property)* on page 60)
- CookiePath (see *CookiePath (property)* on page 60)

CookieExpiration (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

When set to `true`, the client checks if the cookie expiration matches the system time during GET/POST. Use this property if you need to emulate the setting of client side cookies or modify server cookies on the client side.

Note: This property can only be inserted manually.

Example

```
wlGlobals.CookieExpiration = false
```

See also

- CookieDomain (see *CookieDomain (property)* on page 59)
- CookiePath (see *CookiePath (property)* on page 60)

CookiePath (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

When set to `true`, the client checks if the cookie path matches the request path during GET/POST. Use this property if you need to emulate the setting of client side cookies or modify server cookies on the client side.

Note: This property can only be inserted manually.

Example

```
wlGlobals.CookiePath = false
```

See also

- CookieDomain (see CookieDomain (property) on page 59)
- CookieExpiration (see *CookieExpiration (property)* on page 60)

CopyFile() (function)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`SrcFileName`	A literal string or variable containing the full literal name of the file to be copied. WebLOAD assumes that the source file is located in the default directory specified in the File Locations tab (User Copy Files entry) in the **Tools ➤ Global Options** dialog box in the WebLOAD Console or in the **Tools ➤ Settings** dialog box in the WebLOAD Recorder. For additional information about the file's location, refer to *Determining the Included File Location* in the *WebLOAD Scripting Guide*.
`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 WebLOAD 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 Script. For example:

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

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

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

GUI mode

Note: `CopyFile()` and `IncludeFile()` functions can be added directly to the code in a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

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.

The load engine first looks for the file to be copied in the default User Copy Files directory. If the file is not there, the file request is handed over to WebLOAD, which searches for the file using the following search path order:

1. If a full path name has been hardcoded into the CopyFile command, the system searches the specified location. If the file is not found in an explicitly coded directory, the system returns an error code of File Not Found and will not search in any other locations.

Note: It is not recommended to hardcode a full path name, since the Script will then not be portable between different systems. This is especially important for networks that use both UNIX and Windows systems.

2. Assuming no hardcoded full path name in the Script code, the system looks for the file in the current working directory, the directory from which WebLOAD was originally executed.
3. Finally, if the file is still not found, the system searches for the file sequentially through all the directories listed in the File Locations tab.

See also

- Close() (see *Close() (function)* on page 52)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile (see *wlOutputFile (object)* on page 323)
- wlOutputFile() (see *wlOutputFile (object)* on page 323)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

CreateDOM() (function)

Description

`CreateDOM` functions return a complete Document Object Model (DOM) tree. You may compare this expected DOM to the actual DOM generated automatically as your JavaScript Script runs.

Note: WebLOAD uses an extended version of the standard DOM. For more information, see *Understanding the WebLOAD DOM structure* in the *WebLOAD Scripting Guide*.

Syntax

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

Parameters

Parameter Name	Description
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

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)

CreateTable() (function)

Description

WebLOAD provides a `CreateTable` function to automatically convert the tables found on an HTML page to parallel `wlTables` 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 `wlTables` 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 `wlTables` data.

Syntax

```
CreateTable(HTMLFileName)
```

Parameters

Parameter Name	Description
`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.

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)

- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)
- wlTables (see *wlTables (object)* on page 333)

Data (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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`.

Data is for posting data that is not meant to be HTTP encoded, for example Web service calls.

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() (see *wlClear() (method)* on page 301)

Properties

- type (see *type (property)* on page 288)
- value (see *value (property)* on page 294)

Comment

`Data` and `DataFile` are both collections that hold sets of data. `Data` collections are stored within the Script 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 Script code itself. When working with `DataFile` collections, only the name of the text file is stored in the Script itself.

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

See also

- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)

DataFile (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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() (see *wlClear() (method)* on page 301)

Properties

- FileName (see *FileName (property)* on page 93)

Comment

`DataFile` is used for sending files and parallels the posting of mulipart data in HTML. `Data` and `DataFile` are both collections that hold sets of data. `Data` collections are stored within the Script 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 Script code itself, or binary data. When working with `DataFile` collections, only the name of the text file is stored in the Script itself.

See also

- Data (see *Data (property)* on page 66)
- Erase (see *Erase (property)* on page 88)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- value (see *value (property)* on page 294)

DebugMessage() (function)

Description

Displays a debug message in the Log View of WebLOAD Recorder.

Syntax

```
DebugMessage(msg)
```

Parameters

Parameter Name	Description
`msg`	A string with an informative message to be sent to WebLOAD Recorder, to be displayed in the Log View.

Comment

If you call `DebugMessage()` in the main script, WebLOAD sends a debug message to the Log View of WebLOAD Recorder. The message is not written to the Console's Log

View during script execution and has no impact on the continued execution of the Script.

GUI mode

WebLOAD recommends adding message functions to your Script files directly through the WebLOAD Recorder. Drag the **Message** icon from the General toolbox into the Script Tree at the desired location.

See also

- InfoMessage() (see *InfoMessage() (function)* on page 153)

DecodeBinaryEnd (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The DecodeBinaryEnd property is used in conjunction with SaveSource and document.wlSource to enable the user to parse binary data returned from the server. SaveSource and document.wlSource are used to store the server response in wlSource. Since the JavaScript engine does not know how to represent NULLs (binary 0), the engine will convert all binary nulls in the response body to the value of the DecodeBinaryNullAs string. The DecodeBinaryEnd and DecodeBinaryStart properties are used to limit this action to a specific range in the response buffer. If they are not set, the engine will search for binary nulls in the entire buffer. DecodeBinaryEnd and DecodeBinaryStart are used as performance safeguards in case the buffer is very large and you want to parse a section at the start of the buffer.

The value of DecodeBinaryEnd starts from 0 and designates an offset from the beginning of the buffer. The default value of DecodeBinaryEnd is **-1**. This indicates that starting from the DecodeBinaryStart location until the end of the buffer will be converted to binary nulls.

Note: This property can only be inserted manually.

Example

```
wlGlobals.DecodeBinaryEnd=4
```

See also

- DecodeBinaryNullAs (see *DecodeBinaryNullAs (property)* on page 70)

- DecodeBinaryStart (see *DecodeBinaryStart (property)* on page 70)

DecodeBinaryNullAs (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Supports the decoding of binary data. Decoding is not performed by default. In order to decode binary data, the user must call DecodeBinaryNullAs and provide a string value to replace the NULL character.

Note: This property can only be inserted manually.

Syntax

```
wlGlobals.DecodeBinaryNullAs = "TextString"
```

Example

```
WLGlobals.DecodeBinaryNullAs = "Classified"
```

See also

- DecodeBinaryEnd (see *DecodeBinaryEnd (property)* on page 69)
- DecodeBinaryStart (see *DecodeBinaryStart (property)* on page 70)

DecodeBinaryStart (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The DecodeBinaryStart property is used in conjunction with SaveSource and document.wlSource to enable the user to parse binary data returned from the server. SaveSource and document.wlSource are used to store the server response in wlSource. Since the JavaScript engine does not know how to represent NULLs (binary 0), the engine will convert all binary nulls in the response body to the value of the DecodeBinaryNullAs string. The DecodeBinaryEnd and DecodeBinaryStart properties

are used to limit this action to a specific range in the response buffer. If they are not set, the engine will search for binary nulls in the entire buffer. DecodeBinaryEnd and DecodeBinaryStart are used as performance safeguards in case the buffer is very large and you want to parse a section at the start of the buffer.

The value of DecodeBinaryEnd starts from 0 and designates an offset from the beginning of the buffer. The default value of DecodeBinaryStart is **-1**. This indicates that starting from the beginning of the buffer until the DecodeBinaryEnd location will be converted to binary nulls.

Note: This property can only be inserted manually.

Example

```
wlGlobals.DecodeBinaryStart=1
```

See also

- DecodeBinaryEnd (see *DecodeBinaryEnd (property)* on page 69)
- DecodeBinaryNullAs (see *DecodeBinaryNullAs (property)* on page 70)

defaultchecked (property)

Property of Object

- element (see *element (object)* on page 80)

Description

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

See also

- checked (see *checked (property)* on page 48)
- cols (see *cols (property)* on page 54)
- defaultvalue (see *defaultvalue (property)* on page 72)
- id (see *id (property)* on page 146)
- InnerText (see *InnerText (property)* on page 156)
- MaxLength (see *MaxLength (property)* on page 170)
- Name (see *Name (property)* on page 174)
- option (see *option (object)* on page 185)
- row (see *row (object)* on page 223)
- selectedindex (see *selectedindex (property)* on page 235)

- Size (see *Size (property)* on page 247)
- title (see *title (property)* on page 284)
- type (see *type (property)* on page 288)
- Url (see *Url (property)* on page 289)
- value (see *value (property)* on page 294)

defaultselected (property)

Property of Object

- option (see *option (object)* on page 185)

Description

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

See also

- defaultchecked (see *defaultchecked (property)* on page 71)
- selected (see *selected (property)* on page 235)
- value (see *value (property)* on page 294)

defaultvalue (property)

Property of Object

- element (see *element (object)* on page 80)

Description

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

DefineConcurrent() (function)

Description

Use the `DefineConcurrent()` function to define the beginning point, after which all Post and Get HTTP requests are collected, but not executed, until an `ExecuteConcurrent()` function is run. At this point, the collected HTTP requests are executed concurrently, by two or more threads. The number of threads is defined

in WebLOAD Console in the multithreading number in the Browser Parameters tab of the Script Options dialog box.

To simultaneously execute Post and Get HTTP requests, you must define where in the script to begin collecting the requests and where to stop collecting and begin executing them. The HTTP requests are collected until the engine encounters an `ExecuteConcurrent()` function in the script. For more information about the `ExecuteConcurrent()` function, see *ExecuteConcurrent() (function)* (on page 92).

All requests performed from the beginning of the `DefineConcurrent()` function to the `ExecuteConcurrent()` function are stored in an array of documents. You can access every document by index number or document name as follows:

- By index: `wlConcurrentDocuments[i]`
- By DocName: `wlConcurrentDocuments["documentname"]`
  The DocName is an optional name you set for a document for quick access from `wlConcurrentDocuments`. The format for setting the name is:
  `wlHttp.DocName = "documentname"`
  where `DocName` is written with a capital `D` and `N`.
  The default document name is: `all_Concurrent_<index>`.

Example

```
DefineConcurrent()
…
  <any valid JavaScript code, including Post and Get requests>
wlHttp.DocName = "document"


…
ExecuteConcurrent()
for (i=0;i<wlConcurrentDocuments.length;i++)
  InfoMessage("index "+i+" : "+wlConcurrentDocuments[i].URL)
InfoMessage("by DocName: "+wlConcurrentDocuments["document"].URL)
InfoMessage("default name: "
+wlConcurrentDocuments["all_Concurrent_2"].URL)
```

GUI mode

Note: The `DefineConcurrent()` function is usually inserted into script files directly through the WebLOAD Recorder. Drag the **Define Concurrent** icon from the Load toolbox into the Script Tree at the desired location.

For additional information about the `DefineConcurrent()` function, refer to *Define Concurrent* in the *WebLOAD Recorder User's Guide*.

See also

- ExecuteConcurrent() (see *ExecuteConcurrent() (function)* on page 92)

Delete() (method)

Delete() (HTTP method)

Method of Objects

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

- wlHttp (see *wlHttp (object)* on page 316)

Description

Perform an HTTP or HTTPS Delete command.

Syntax

```
Delete([URL]
```

Parameters

Parameter Name	Description
`[URL]`	An optional parameter identifying the document URL. You may optionally specify the URL as a parameter of the method. `Delete()` connects to the first URL that has been specified from the following list, in the order specified: • A `Url` parameter specified in the method call. • The `Url` property of the `wlHttp` object. • The local default `wlLocals.Url`. • The global default `wlGlobals.Url`.

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Head() (see *Head() (method)* on page 139)
- Header (see *Header (property)* on page 140)

- Options() (see *Options() (method)* on page 186)
- Post() (see *Post() (method)* on page 205)
- Put() (see *Put() (method)* on page 213)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

Delete() (cookie method)

Method of Objects

- wlCookie (see *wlCookie (object)* on page 302)

Description

This method deletes all cookies set by `wlCookie` in the current thread.

Syntax

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

Parameters

Parameter Name	Description
`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, "`/`".

Example

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

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)

- *Using the IntelliSense JavaScript Editor* (on page 18)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

DeleteEmptyCookies (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Indicates whether to delete existing cookies if the server sends a "set-cookie" header with empty values for the existing cookies. If DeleteEmptyCookies is false, the existing cookies are set to their empty value (null). The default value of DeleteEmptyCookies is **false**.

Example

```
wlGlobals.DeleteEmptyCookies = false
```

Comments

Some servers tell the client to delete existing cookies by sending the client empty cookies.

DisableSleep (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Setting this property defines how the engine should handle the Sleep command in the script. This boolean flag indicates whether the recorded sleep pauses will be included in the test session (false) or ignored (true).

Example

```
wlGlobals.DisableSleep = true
```

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 WebLOAD Console. Select a setting from the Sleep Time Control tab of the **Default**, **Current** or **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- Sleep() (see *Sleep() (function)* on page 248)
- SleepDeviation (see *SleepDeviation (property)* on page 249)
- SleepRandomMax (see *SleepRandomMax (property)* on page 250)
- SleepRandomMin (see *SleepRandomMin (property)* on page 251)

DNSUseCache (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

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

- **false** – Disable IP address caching.
- **true** – Enable IP address caching (default).

Assign a `true` value to reduce the time for domain name resolution. Assign a `false` value if you want to include the time for name resolution in the WebLOAD performance statistics.

GUI mode

WebLOAD recommends enabling or disabling the DNS cache through the WebLOAD Console. Enable caching for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

To clear the DNS cache, set the ClearDNSCache() (see *ClearDNSCache() (method)* on page 49) property.

See also

- *HTTP Components* (on page 24)
- ClearDNSCache() (see *ClearDNSCache() (method)* on page 49)
- ClearSSLCache() (see *ClearSSLCache() (method)* on page 49)
- SSLUseCache (see *SSLUseCache (property)* on page 272)

document (object)

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 a script. 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, to access links, use the following expression:

```
document.links[0]
```

Methods

- wlGetAllForms() (see *wlGetAllForms() (method)* on page 311)
- wlGetAllFrames() (see *wlGetAllFrames() (method)* on page 312)
- wlGetAllLinks() (see *wlGetAllLinks() (method)* on page 312)

Properties

- form (see *form (object)* on page 95)
- frames (see *frames (object)* on page 99)
- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)
- script (see *script (object)* on page 228)
- title (see *title (property)* on page 284)
- wlHeaders (see *wlHeaders (object)* on page 314)
- wlMetas (see *wlMetas (object)* on page 320)
- wlSource (see *wlSource (property)* on page 330)
- wlStatusLine (see *wlStatusLine (property)* on page 331)
- wlStatusNumber (see *wlStatusNumber (property)* on page 331)
- wlTables (see *wlTables (object)* on page 333)
- wlVersion (see *wlVersion (property)* on page 338)
- wlXmls (see *wlXmls (object)* on page 340)

ElapsedRoundTime (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

The minimum amount of time (in milliseconds) for the round to be played back. If the total time it takes for the round to be played back is less than the time period specified, the machine sleeps for the remainder of the time. This property must be set in `InitAgenda()`. If it is set anywhere else, it is ignored.

The behavior of the sleep time is affected by the Sleep Time Control settings that are set in the Current Project Options of the WebLOAD Recorder and Console. These settings can be one of the following:

- **Sleep time as recorded (default for the Console)** – Runs the script with the delays corresponding to the natural pauses that occurred when recording the script.
- **Ignore recorded sleep time (default for the WebLOAD Recorder)** – Eliminates any pauses when running the script and runs a worst-case stress test.
- **Set random sleep time** – Sets the ranges of delays to represent a range of users.
- **Set sleep time deviation** – Sets the percentage of deviation from the recorded value to represent a range of users.

For more information on setting the Sleep Time Control settings, see *Configuring Sleep Time Control Options* in the *WebLOAD Recorder User's Guide*.

Example

```
function InitAgenda()
{
  wlGlobals.ElapsedRoundTime = 1056
}
```

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 (see *form (object)* on page 95)

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 (see *form (object)* on page 95) 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
```

Properties

- checked (see *checked (property)* on page 48)
- cols (see *cols (property)* on page 54)
- defaultchecked (see *defaultchecked (property)* on page 71)
- defaultvalue (see *defaultvalue (property)* on page 72)
- InnerText (see *InnerText (property)* on page 156)
- Name (see *Name (property)* on page 174)
- id (see *id (property)* on page 146)
- InnerImage (see *InnerImage (property)* on page 155)
- InnerText (see *InnerText (property)* on page 156)
- MaxLength (see *MaxLength (property)* on page 170)
- option (see *option (object)* on page 185)
- OuterLink (see *OuterLink (property)* on page 186)
- row (see *row (object)* on page 223)
- selectedindex (see *selectedindex (property)* on page 235)
- Size (see *Size (property)* on page 247)
- title (see *title (property)* on page 284)

- type (see *type (property)* on page 288)
- Url (see *Url (property)* on page 289)
- value (see *value (property)* on page 294)

Comment

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

See also

- *Collections* (on page 27)
- Image (see *Image (object)* on page 149)
- *Select* (on page 230)

EncodeBinary (property)

The EncodeBinary property is identical to the EncodeRequestBinaryData property. For additional information, see *EncodeRequestBinaryData (property)* on page 83.

EncodeFormdata (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Sets the wlGlobals.EncodeFormdata flag.

Generally, when an HTTP client (Microsoft Internet Explorer/Firefox or WebLOAD) sends a post request to the server, the data must be HTTP encoded. Special characters such as blanks, “>“ signs, and so on, are replaced by “%xx”. For example, a space is encoded as “%20”.

Turn off the encoding when the script sends large requests that have no data that needs to be encoded. This improves performance as it bypasses the scanning and reformatting of the request buffer.

GUI mode

In WebLOAD Console, select or deselect the **Encode Form Data** checkbox in the HTTP Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, select or deselect the **Encode Form Data** checkbox in the HTTP Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Example

wlGlobals.EncodeFormData = true

See also

- *HTTP Components* (on page 24)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

EncodeRequestBinaryData (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

Used to specify if the binary data in requests should be encoded. The default value of EncodeRequestBinaryData is **false**.

For example, if a mobile operator wants to simulate the sending of binary data from the browser (phone) to the server. Part of the binary data is a value (for example, phone number) that needs parameterization. When the EncodeRequestBinaryData flag is set to true, the binary form data "x0Ax0BAMIRx00" appears as "%0A%0BAMIR%00" in the script.

Example

```
wlGlobals.EncodeRequestBinaryData = true
```

GUI mode

In WebLOAD Recorder, check **Encode Binary Data** in the Script Generation tab of the **Recording and Script Generation Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- EncodeResponseBinaryData (see *EncodeResponseBinaryData (property)* on page 84)
- EncodeBinary (see *EncodeBinary (property)* on page 82)
- SaveSource (see *SaveSource (property)* on page 226)

EncodeResponseBinaryData (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

Indicates whether binary data sent in responses should be encoded. EncodeResponseBinaryData can be used with web pages that have binary data sent in responses and on which you would want to perform correlation on that binary data.

The default value of EncodeResponseBinaryData is **false**. When set to true, the response will be encoded when the user accesses document.wlSource. The encoding is performed on the original data, when it is accessed. Readable characters that are not letters are not encoded. That is, "!@#$%^&*()" remains "!@#$%^&*()" and carriage return and tab are translated to \r\t. The response is saved in document.wlSource only if the SaveSource flag is set to true.

Example

```
wlGlobals.EncodeResponseBinaryData = true
```

See also

- EncodeRequestBinaryData (see *EncodeRequestBinaryData (property)* on page 83)

encoding (property)

Property of Object

- form (see *form (object)* on page 95)

Description

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

See also

- form (see *form (object)* on page 95)

EndTransaction() (function)

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>
  …
  [SetFailureReason(ReasonName)]
EndTransaction(TransName,Verification,[SaveFlag],[FailureReason])
```

Parameters

Parameter Name	Description
`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) (see *VerificationFunction() (user-defined) (function)* on page 297), 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).
`[FailureReason]`	An optional user-supplied string that provides a reason for the failure.

GUI mode

Note: `BeginTransaction()` and `EndTransaction()` functions are usually accessed and inserted into script files directly through the WebLOAD Recorder. For example, the following figure illustrates a section in the Script Tree bracketed by BeginTransaction and EndTransaction nodes. The EndTransaction node is highlighted in the Script Tree.

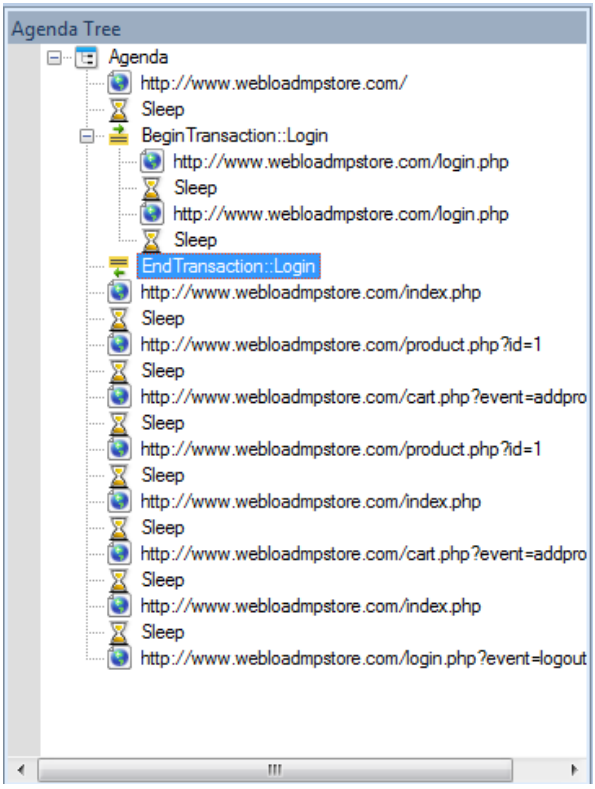

Figure 9: Form Branch in Script Tree Bracketed by BeginTransaction and EndTransaction Nodes

To mark the end of a transaction, drag the **End Transaction** icon from the Load toolbox into the Script Tree, directly after the last action you want included in the script.

For additional information about the `EndTransaction()` function, refer to *Begin and End Transaction* in the *WebLOAD Recorder User's Guide*.

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

EnforceCharEncoding (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Indicates whether the parser should use the character set it parses in the HTML pages or override it using the `CharEncoding` property. The default value is **false** (use the encoding from the HTML pages).

The EnforceCharEncoding property can be set to one of the following values:

- **true** – Use the `CharEncoding` property.
- **false** (default) – Get the encoding from the HTML pages.

Example

```
wlGlobals.EnforceCharEncoding = false
```

GUI mode

In WebLOAD Console, check **Enforce Character Encoding** in the Browser Parameters tab of the **Default Options** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, check **Enforce Character Encoding** in the Browser Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- CharEncoding (see *CharEncoding (property)* on page 47)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Erase (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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 script is incorrect. In this script, the `wlHttp` properties are assigned values in `InitClient()`. The script 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 script, 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 (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- DataCollection.type (see *type (property)* on page 288)
- DataCollection.value (see *value (property)* on page 294)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- wlClear() (see *wlClear() (method)* on page 301)

ErrorMessage() (function)

Description

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

Syntax

```
ErrorMessage(msg)
```

Parameters

Parameter Name	Description
`msg`	A string with an error message to be sent to the WebLOAD Console.

Comment

If you call `ErrorMessage()` in the main script, WebLOAD stops the current round of execution. 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 script.

GUI mode

WebLOAD recommends adding message functions to your script files directly through the WebLOAD Recorder. Message function commands can be added to the script in Visual Editing mode using the Toolbox message item and the Insert menu command.

Message function command lines may also be added directly to the code in a JavaScript Object within a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

See also

- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)

- wlException (see *wlException (object)* on page 306)
- wlException() (see *wlException() (constructor)* on page 308)

ErrorMessage (property)

Property of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

`ErrorMessage` is used to define a global error message that appears in the Log window when a verification fail error occurs. When defined, `ErrorMessage` affects all the verifications in which an error message is not defined. If you define an error message for a specific verification, it overrides the global error message defined in the `ErrorMessage` property.

Example

To set the global error message displayed in the Log window in the event of any verification fail errors to `my personalized error message`, write:

```
wlVerification.ErrorMessage = "my personalized error message"
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- PageTime (see *PageTime (property)* on page 190)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- Title (see *Title (function)* on page 285)

EvaluateScript() (function)

Description

Enables testers to include scripts and specify the point during script execution at which the script should be executed.

Syntax

```
EvaluateScript("Script", RunModeConstant)
```

Parameters

Parameter Name	Description
Script	A valid JavaScript syntax, including function calls.
RunModeConstant	One of the following list of constants that acts as a flag when passed as a parameter to EvaluateScript(). Defines the point during script execution at which WebLOAD should execute the script being included here. Possible choices include: • WLAfterInitAgenda • WLBeforeInitClient • WLBeforeThreadActivation • WLOnThreadActivation • WLBeforeRound • WLAfterRound • WLAfterTerminateClient • WLAfterTerminateAgenda

Comment

If the script to be executed is in an external file, use the following:

```
IncludeFile(filename.js)
EvaluateScript("MyFunction()",WLAfterRound)
```

Where `MyFunction()` is defined in filename.js.

event (property)

Property of Objects

- link (see *link (object)* on page 162)
- script (see *script (object)* on page 228)

Description

Represents the event that occurred to the parent object or the event for which the script is written.

ExecuteConcurrent() (function)

Description

Use the `ExecuteConcurrent()` function to define the point after which all Post and Get HTTP requests, which have been collected since the `DefineConcurrent()` function was run, are executed. At this point, the collected HTTP requests are executed concurrently, by two or more threads. The number of threads is defined in the

WebLOAD Console in the multithreading number in the Browser Parameters tab of the Script Options dialog box.

Note: This function can only be inserted in your script *after* a `DefineConcurrent()` function. For more information about the `DefineConcurrent()` function, see *DefineConcurrent() (function)* (on page 72).

When the engine encounters the `ExecuteConcurrent()` function, it stops collecting the HTTP requests in the script and starts their execution.

Example

```
DefineConcurrent()
…
  <any valid JavaScript code, including Post and Get requests>
…
ExecuteConcurrent()
```

GUI mode

Note: The `ExecuteConcurrent()` function is usually inserted into script files directly through the WebLOAD Recorder. Drag the **Execute Concurrent** icon, from the Load toolbox, into the Script Tree at the desired location.

For additional information about the `ExecuteConcurrent()` function, refer to *Execute Concurrent* in the *WebLOAD Recorder User's Guide*.

See also

- DefineConcurrent() (see *DefineConcurrent() (function)* on page 72)

extractValue()(function)

Description

Use this function to extract a specific string contained within another string.

Syntax

```
retVarName = extractValue(prefix, suffix, str, instance)
```

Parameters

Parameter Name	Description
`retVarName`	A variable name that will be generated to the agenda
`prefix`	A string indicating the beginning of the string to be extracted.
`suffix`	A string indicating the end of the string to be extracted.

Parameter Name	Description
str	The string to be extracted is contained within this string.
instance	When there is more than one appearance of the prefix string following by the suffix string, this optional parameter can be used to indicate the correct string to be returned. The default value is 1. For example, when instance is 3, the third appearance of the prefix string followed by the suffix string indicates the string to be returned.

Return Value

The extractValue function returns the extracted string.

Example

The following function extracts 'x' out of 'axb':

```
retStr = extractValue("a", "b", "axb")
```

Since no instance parameter is specified, WebLOAD automatically adds the default value of the instance parameter:

```
retStr = extractValue("a", "b", "axb",1)
```

The following function extracts 'tttatt' out of 'zzzatttattbaxbzzzbzz':

```
retStr = extractValue("a", "b","zzzatttattbaxbzzzbzz",1)
```

The following function extracts 'x' out of 'zzzatttattbaxbzzzbzz':

```
retStr = extractValue("a", "b","zzzatttattbaxbzzzbzz",2)
```

FileName (property)

Property of Object

- wlHttp.DataFile (see *DataFile (property)* on page 67)

Description

This property is a string that holds the name of the file being submitted through an HTTP Post command.

Syntax

```
wlHttp.DataFile.FileName = "DataFileName"
```

See also

- Data (see *Data (property)* on page 66)

- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- type (see *type (property)* on page 288)
- value (see *value (property)* on page 294)
- wlClear() (see *wlClear() (method)* on page 301)
- wlHttp (see *wlHttp (object)* on page 316)

FilterURL (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The value of the FilterURL property is a list of filters separated by semi-colons. When retrieving a resource, the engine checks whether the value of any of these filters appear in the URL. If the value of any of the filters appears in the URL, the URL is not executed. Filtering is only performed during playback.

Example

For example, if `FilterURL = "ynet;cnn.com"`, the engine will filter URLs from ynet.com and ynet.co.il, as well as URLS from cnn.com.

See also

- *HTTP Components* (on page 24)

form (object)

Property of Object

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

- document (see *document (object)* on page 78)

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 (see *element (object)* on page 80) 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 (see *Collections* on page 27). 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>
```

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
```

Properties

- element (see *element (object)* on page 80)
- encoding (see *encoding (property)* on page 83)
- id (see *id (property)* on page 146)
- method (see *method (property)* on page 171)
- Name (see *Name (property)* on page 174)
- target (see *target (property)* on page 280)
- Url (see *Url (property)* on page 289)

See also

- *Collections* (on page 27)
- document (see *document (object)* on page 78)
- element (see *element (object)* on page 80)
- Image (see *Image (object)* on page 149)
- *Select* (on page 230)

FormData (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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. FormData goes through HTTP encoding before being sent to the server in the same manner as content-type=application/x-www-form-urlencoded.

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() (see *wlClear() (method)* on page 301) method 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 also

- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- Get() (see *Get() (transaction method)* on page 104)

- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- type (see *type (property)* on page 288)
- value (see *value (property)* on page 294)
- wlClear() (see *wlClear() (method)* on page 301)

frames (object)

Property of Object

- document (see *document (object)* on page 78)

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 a script. 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>
```

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

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

-Or-

```
document.frames.namestring
```

Comment

If the GetFrames property is `false`, the frames collection is empty.

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 (see *id (property)* on page 146)
- Index (see *Index (property)* on page 152)
- Name (see *Name (property)* on page 174)
- title (see *title (property)* on page 284)
- Url (see *Url (property)* on page 289)

See also

- *Collections* (on page 27)
- GetFrames (see *GetFrames (property)* on page 117)

Function (property)

Property of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

`Function` is used to define a global JavaScript function called when a verification fail error occurs. When defined, `Function` affects all the verifications in which a JavaScript function is not defined. If you define a JavaScript function for a specific verification, it overrides the global JavaScript function defined in the `Function` property.

Example

To set the global JavaScript function called in the event of any verification fail errors to `GetOperatingSystem()`, write:

```
wlVerification.Function = GetOperatingSystem()
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- PageTime (see *PageTime (property)* on page 190)
- Severity (see *Severity (property)* on page 247)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

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()
```

Return Value

Returns a unique identification string for the current Load Generator.

GUI mode

WebLOAD recommends accessing global system variables, including the `GeneratorName()` identification function through the WebLOAD Recorder. All the variables that appear in this list are available for use at all times in a script file. In the WebLOAD Recorder main window, click **Variables Windows** in the **Debug** tab of the ribbon.

For example, it is convenient to add `GeneratorName()` to a Message Node to clarify which Load Generator sent the messages that appear in the WebLOAD Console Log window.

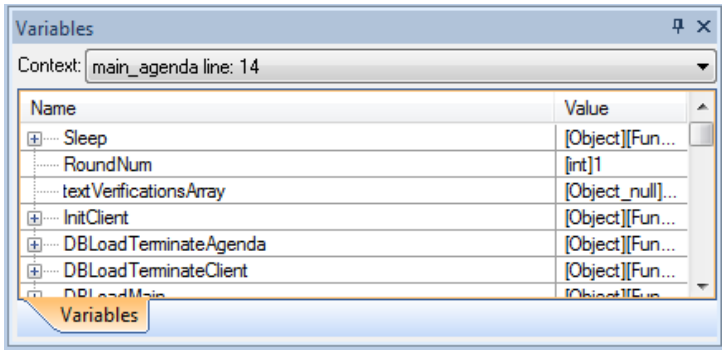

Figure 10: Variables List in WebLOAD Recorder

See also

- ClientNum (see *ClientNum (variable)* on page 50)
- GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131)
- *Identification Variables and Functions* (on page 29)
- RoundNum (see *RoundNum (variable)* on page 222)
- VCUniqueID() (see *VCUniqueID() (function)* on page 296)

Get() (method)

Get() (addition method)

Method of Objects

- wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309)
- wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)

Description

Returns the current value of the specified shared variable.

Syntax

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

Parameters

Parameter Name	Description
SharedVarName	The name of a shared variable whose value should be returned.

Parameter Name	Description
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 script, 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 scripts, 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 script, 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 scripts, 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() (see *Add() (method)* on page 39)
- Set() (see *Set() (addition method)* on page 240)

Get() (cookie method)

Method of Objects

- location (see *location (object)* on page 168)
- wlCookie (see *wlCookie (object)* on page 302)

Description

Searches for the value of a specific cookie and returns it. If there is more than one cookie with the same name, the method returns the first occurrence.

Syntax

```
wlCookie.Get(name[, domain][, path])
```

Parameters

Parameter Name	Description
name	A descriptive name identifying the cookie, for example, "CUSTOMER".
domain	The top-level domain name of the cookie, for example, "www.ABCDEF.com".
path	The top-level directory path, within the specified domain, of the cookie, for example, "/".

Return Value

Returns the value of the cookie found.

Example

```
retValue = wlCookie.Get("CUSTOMER", "www.ABCDEF.com", "/" )
```

Get() (transaction method)

Method of Objects

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

- wlHttp (see *wlHttp (object)* on page 316)

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

Parameter Name	Description
`[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, in the order specified: • A `Url` parameter specified in the method call. • The `Url` property of the `wlHttp` object. • The local default `wlLocals.Url`. • The global default `wlGlobals.Url`.
`[TransName]`	An optional user-supplied string with the transaction name as it will appear in the Statistics Report. 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. In the WebLOAD 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 script. The name assigned by WebLOAD is simply the URL used by that Get or Post transaction. If your script includes multiple transactions to the same URL, the information will be collected cumulatively for those transactions.

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:
url= http://www.ABCDEF.com/product_info.html
wlHttp.Get(url,
                        "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 script'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() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Delete() (see *Delete() (HTTP method)*) on page 74

- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Head() (see *Head() (method)* on page 139)
- Header (see *Header (property)* on page 140)
- Options() (see *Options() (method)* on page 186)
- Post() (see *Post() (method)* on page 205)
- Put() (see *Put() (method)* onpage 213)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

GetApplets (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of Java Applets in an HTML page. The default value of GetApplets is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetApplets = true
```

See also

- GetCss() (see *GetCss (property)* on page 108)
- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)

- GetXml() (see *GetXml() (property)* on page 138)

GetCss (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of cascading style sheets in an HTML page. The default value of GetCss is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetCss = true
```

See also

- GetApplets() (see *GetApplets (property) on* page 107)
- GetEmbeds() (see *GetEmbeds (property) on* page 113)
- GetFrames() (see *GetFrames (property) on* page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)
- GetXml() (see *GetXml() (property)* on page 138)

GetElementById() (function)

Description

Used to retrieve the element with the specified identification value by querying the DOM of the HTML from the last response.

Syntax

```
GetElementById("id")
```

Parameters

Parameter Name	Description
id	The identification value of the element to retrieve, enclosed in quotation marks.

Return Value

The first element with the requested identification value or Null if no element was found.

See also

- GetElementByName() (see *GetElementByName() (function)* on page 110)
- GetElementsById() (see *GetElementsById() (function)* on page 109)

GetElementsById() (function)

Description

Used to retrieve an array of all elements with the specified identification value by querying the DOM of the HTML from the last response.

Note: An element can be from the `document.forms[].elements[]`, `document.links[]` or `document.images[]` collections.

Syntax

```
GetElementsById("id")
```

Parameters

Parameter Name	Description
id	The identification value of the elements to retrieve, enclosed in quotation marks.

Return Value

A list of the requested elements.

Example

```
wlHttp.Get("www.abc.com")
var elementArr = GetElementsById("id4");
for (var i in elementArr ) {
  var elm = elementArr[i];
  InfoMessage( "ID:" + elm.id + ", Name:" + elm.name + ", Type:" +
  elm.type + ", Value:" + elm.value );
}
```

The expected output is:

```
4.11        ID:id4, Name:event, Type:hidden, Value:search
4.23        ID:id4, Name:process, Type:hidden, Value:login
```

See also

- GetElementsByName() (see *GetElementsByName() (function)* on page 110)
- GetElementById() (see *GetElementById() (function)* on page 108)

GetElementByName() (function)

Description

Used to retrieve the element with the specified name by querying the DOM of the HTML from the last response.

Note: An element can be from the document.forms[].elements[], document.links[] or document.images[] collections.

Syntax

```
GetElementByName("name")
```

Parameters

Parameter Name	Description
name	The name of the element to retrieve, enclosed in quotation marks.

Return Value

The first element with the requested name or Null if no element was found.

See also

- GetElementsByName() (see *GetElementsByName() (function)* on page 110)
- GetElementById() (see *GetElementById() (function)* on page 108)

GetElementsByName() (function)

Description

Used to retrieve an array of all elements with the specified name by querying the DOM of the HTML from the last response.

Note: An element can be from the document.forms[].elements[], document.links[] or document.images[] collections.

Syntax

```
GetElementsByName("name")
```

Parameters

Parameter Name	Description
name	The name of the elements to retrieve, enclosed in quotation marks.

Return Value

A list of the requested elements.

Example

```
wlHttp.Get("http://www.webloadmpstore.com/login.php")
var elementArr = GetElementsByName("event");
for (var i in elementArr ) {
  var elm = elementArr[i];
  InfoMessage( "Name:" + elm.name + ", ID:" + elm.id + ", Type:" +
  elm.type + ", Value:" + elm.value );
}
```

The expected output is:

```
4.11        Name:event, ID:, Type:hidden, Value:search
4.23        Name:event, ID:, Type:hidden, Value:login
```

See also

- GetElementByName() (see *GetElementByName() (function)* on page 110)
- GetElementsById() (see *GetElementsById() (function)* on page 109)

GetElementValueById() (function)

Description

Used to retrieve the value of the element with the specified identification value by querying the DOM of the HTML from the last response.

Note: An element can be from the document.forms[].elements[], document.links[] or document.images[] collections.

Syntax

```
GetElementValueById("id")
```

Parameters

Parameter Name	Description
id	The identification value of the element, enclosed in quotation marks.

Return Value

The value of the first element with the requested identification value or Null if no element was found.

Example

```
GetElementValueById("sessionid")
```

See also

- GetElementValueByName() (see *GetElementValueByName() (function)* on page 112)

GetElementValueByName() (function)

Description

Used to retrieve the value of the element with the specified name by querying the DOM of the HTML from the last response.

Note: An element can be from the document.forms[].elements[], document.links[] or document.images[] collections.

Syntax

```
GetElementValueByName("name")
```

Parameters

Parameter Name	Description
name	The name of the element, enclosed in quotation marks.

Return Value

The value of the first element with the requested name or Null if no element was found.

Example

```
wlHttp.Get("http://www.webloadmpstore.com/login.php")
var elementArr = GetElementValueByName("event");
for (var i in elementArr ) {
  var elm = elementArr[i];
  InfoMessage( "Name:" + elm.name + ", ID:" + elm.id + ", Type:" +
  elm.type + ", Value:" + elm.value );
}
```

See also

- GetElementValueById() (see *GetElementValueById() (function)* on page 111)

GetEmbeds (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of embedded objects in an HTML page. The default value of GetEmbeds is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetEmbeds = true
```

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)
- GetXml() (see *GetXml() (property)* on page 138)

GetFieldValue() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`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.

GetFieldValueInForm() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

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

Parameter Name	Description
`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.

```
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.

GetFormAction() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

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

Parameter Name	Description
`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

```
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.

GetFrameByUrl() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve a frame object given its URL.

Syntax

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

Parameters

Parameter Name	Description
`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

```
//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.

Comment out `GetFrames=false` when you use the GetFrameByUrl method.

GetFrames (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of Frames and IFrames in an HTML page. The default value of GetFrames is **true**.

Note: This property can only be inserted manually.

Note: Although the default value for GetFrames is true, during recording, the following is automatically inserted in the script:

```
wlGlobals.GetFrames=false;
```

Example

```
wlGlobals.GetFrames = true
```

Comments

When GetMetas is true, GetFrames should also be true as the redirection is retrieved as a frame (see *GetMetas (property)* on page 130).

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)
- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)
- GetXml() (see *GetXml() (property)* on page 138)

GetFrameUrl() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

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

Comment

Comment out `GetFrames=false` when you use the GetFrameByUrl method.

Return Value

The requested location object.

Comment

This method is equivalent to the location property of a frame object (see *frames (object)* on page 99).

GetHeaderValue() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve the value of an HTTP header field.

Syntax

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

Parameters

Parameter Name	Description
`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".
```

-Or-

```
document.wlHeaders["host"]
document.frame[0].wlHeaders["host"]
```

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.

If you are specifying a frame, comment out `GetFrames=false`.

GetHost() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`[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`".

-Or-

```
document.wlHeaders["hostname"]
document.frame[0].wlHeaders["hostname"]
```

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.

If you are specifying a frame, comment out `GetFrames=false`.

GetHostName() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`[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.

If you are specifying a frame, comment out `GetFrames=false`.

GetImages (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of images in an HTML page. The default value of `GetImages` is **true**.

When `GetImages` is false, the load engine does not retrieve the images from an HTML page.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetImages = true
```

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)

- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)
- GetXml() (see *GetXml() (property)* on page 138)

GetImagesInThinClient (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

When set to `true`, the "Thin" client will retrieve images. The default value of GetImagesInThinClient is **false**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetImagesInThinClient = true
```

See also

- SetClientType (see *SetClientType (function)* on page 242)
- *Collections* (on page 27)
- document (see *document (object)* on page 78)
- Header (see *Header (property)* on page 140)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

GetIPAddress() (method)

Method of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Returns the identity of the *current* IP address.

Syntax

```
GetIPAddress()
```

Return Value

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

Example

```
...
wlHttp.MultiIPSupport = true
CurrentIPAddress = wlHttp.GetIPAddress()
wlHttp.Get()
...
```

Comment

Requesting the identity of the *current* IP address is only meaningful if your script is handling more than one IP address. `GetIPAddress()` therefore can only return a value if `MultiIPSupport=true`. 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 script sets `wlGlobals.MultiIPSupport`, then `GetIPAddress()` returns a value at any point in the script. If you set only `wlHttp.MultiIPSupport`, then `GetIPAddress()` returns a value only if called before the next immediate HTTP transaction.

See also

- *HTTP Components* (on page 24)

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 you opened the file using the default sequential mode (see *Open() (function)* on page 183), 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 any process of the Load Generator (across the master and slave processes of a single Load Generator/script combination) 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 opened the file for random access (see *Open() (function)* on page 183), each successive call in any thread of any process of the Load Generator (across the master and slave processes of a single Load Generator/script combination) reads some randomly selected line of the file. To read the input file lines in random order, you must include `Open(filename, WLRandom)` in the script'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.

Note: The last line of the file should not include a carriage return.

Syntax

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

Parameters

Parameter Name	Description
`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", ":")
```

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() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- *File Management Functions* (on page *28*)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile (see *wlOutputFile() (constructor)* on page 324)
- wlOutputFile() (see *wlOutputFile (object)* on page 323)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

GetLine() (method)

Method of Object

- wlInputFile (see *wlInputFile (object)* on page 317)

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 you opened the file using the default WLFileSequential access method (see *Open() (method)* on page 180), 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 any process of any 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 opened the file using the WLFileSequentialUnique access method (see *Open() (method)* on page 180), then the procedure is basically as when using the WLFileSequential access mode, except that the if the value/row is being used by another VC, it is not retrieved, but skipped.
- If you opened the file using the WLFileRandom access method (see *Open() (method)* on page 180), `GetLine()` reads a random value/row from the file, where there might be multiple access to the same line by different Load Generator machines.
- If you opened the file using the WLFileRandomUnique access method (see *Open() (method)* on page 180), `GetLine()` reads a unique, unused value/row randomly from the file.

Note: The last line of the file should not include a carriage return.

Syntax

```
strInputFileLine = myFileObj.getLine(delimiter)
```

Parameters

Parameter Name	Description
`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 `strInputFileLine` in this example) includes the following elements:

- strInputFileLine [0]-the complete line. For example:

  `"John,Smith, jsmith@ABC.com"`
- strInputFileLine [1]-the first token. In this example:

  `"John"`

- strInputFileLine [2]-the second token. In this example:

  "Smith"
- strInputFileLine [3]-the third token. In this example:

  "jsmith@ABC.com"
- strInputFileLine.LineNum-the number of the line that was just read. For example: 1

Example

To read and parse the next line of the ASCII input file specified in myFileObj:

```
strInputFileLine = GetLine(",")
```

Comment

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 (`strInputFileLine[0]` and `strInputFileLine[1]`), each containing the entire line.

See also

- *File Management Functions* (on page *28*)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlInputFile (see *wlInputFile() (constructor)* on page 318)

GetLinkByName() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`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.

If you are specifying a frame, comment out `GetFrames=false`.

GetLinkByUrl() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

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

Syntax

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

Parameters

Parameter Name	Description
`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.

If you are specifying a frame, comment out `GetFrames=false`.

GetMessage() (method)

Method of Object

- wlException (see *wlException (object)* on page 306)

Description

Returns the message string text stored in this object.

Syntax

```
wlExceptionObject.GetMessage()
```

Return Value

Text string of the error message for this object.

Example

```
MeaningfulErrorMessage = myExceptionObject.GetMessage()
```

See also

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)

- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException (see *wlException (object)* on page 306)

GetMetas (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

The `GetMetas` property, when set to true, enables the support of redirection for non-recorded scripts, for websites using the HTML META tag (for example, www.ynet.co.il).

Note: Since scripts that were recorded automatically include the redirected URL, the `GetMetas` property should be used only in scripts that were written manually and that contain a URL with meta direction.

Example

```
wlGlobals.GetMetas = false
```

Comments

- Because the redirection is retrieved as a frame, the `GetFrames` property must be set to true (see *GetFrames (property)* on page 117).
- The additional wlHttp.GET will not be part of the script (it will be like frame 0).
- The desired page will be requested only on playback.
- The page will not be visible in WebLOAD Recorder's Browser View. This is because redirection will not be performed (the document will not be replaced). WebLOAD implements the redirected URL by adding a frame to the parent HTML. That is, the first page will be added with an extra frame containing the redirection URL (fully parsed and all the objects in it will be get).

GetOperatingSystem() (function)

Description

Returns a string identifying the operating system running on the current Load Generator.

Syntax

```
GetOperatingSystem()
```

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)

See also

- ClientNum (see *ClientNum (variable)* on page 50)
- GeneratorName() (see *GeneratorName() (function)* on page 101)
- *Identification Variables and Functions* (on page 29)
- RoundNum (see *RoundNum (variable)* on page 222)
- VCUniqueID() (see *VCUniqueID() (function)* on page 296)

GetOthers (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of objects not covered by the other Get methods in an HTML page. The default value of GetOthers is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetOthers = true
```

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)
- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetScripts() (see *GetScripts (property)* on page 134)
- GetXml() (see *GetXml() (property)* on page 138)

GetPortNum() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve the port number of the current URL.

Syntax

```
GetPortNum([frame])
```

Parameters

Parameter Name	Description
`[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.

If you are specifying a frame, comment out `GetFrames=false`.

GetQSFieldValue() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

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

Parameter Name	Description
`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".

GetScripts (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of external JavaScript scripts in an HTML page. The default value of GetScripts is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetScripts = true
```

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)
- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetXml() (see *GetXml() (property)* on page 138)

GetSeverity() (method)

Method of Object

- wlException (see *wlException (object)* on page 306)

Description

Returns the severity level value stored in this object.

Syntax

```
wlExceptionObject.GetSeverity()
```

Return Value

Integer, representing one of the following error level values:

- `WLError`-this specific transaction failed and the current test round was aborted. The script 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 script 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

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException() (see *wlException() (constructor)* on page 308)

GetStatusLine() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve the status string from the HTTP header.

Syntax

```
GetStatusLine([frame])
```

Parameters

Parameter Name	Description
[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.

If you are specifying a frame, comment out `GetFrames=false`.

GetStatusNumber() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve the status code from the HTTP header.

Syntax

```
GetStatusNumber([frame])
```

Parameters

Parameter Name	Description
[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.

If you are specifying a frame, comment out `GetFrames=false`.

GetUri() (method)

Method of Object

- wlHtml (see *wlHtml (object)* on page 315)

Description

Retrieve the URI part of a URL. The URI is the portion of the address following the host name.

Syntax

```
GetUri([frame])
```

Parameters

Parameter Name	Description
[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.

If you are specifying a frame, comment out `GetFrames=false`.

GetXML (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables the retrieval of external XML in an HTML page. The default value of GetXML is **true**.

Note: This property can only be inserted manually.

Example

```
wlGlobals.GetXML = true
```

See also

- GetApplets() (see *GetApplets (property)* on page 107)
- GetCss() (see *GetCss (property)* on page 108)
- GetEmbeds() (see *GetEmbeds (property)* on page 113)
- GetFrames() (see *GetFrames (property)* on page 117)
- GetImages() (see *GetImages (property)* on page 121)
- GetOthers() (see *GetOthers (property)* on page 131)
- GetScripts() (see *GetScripts (property)* on page 134)

hash (property)

Property of Object

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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"`.

Head() (method)

Method of Object

- wlHttp (see *wlHttp (object)* on page 316)

Description

Perform an HTTP or HTTPS Head command.

Syntax

```
Head()
```

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

- *HTTP Components* (on page 24)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Post() (see *Post() (method)* on page 205)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlLocals (see *wlLocals (object)* on page 319)

Header (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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 **Default Options** dialog box, accessed from the **Tools** tab of the ribbon. Resetting configuration properties as you run and rerun various testing scenarios enables you to fine tune your tests to match your exact needs at that moment. For example, you can reset the user-agent value through the Browser Parameters tab.

Comment

Use the wlClear() (see *wlClear() (method)* on page 301) method 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 (see *wlCookie (object)* on page 302)

Use the `wlHttp.Header` property to change or reset specific individual values immediately before executing the next `wlHttp GET/POST request`.

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

- *HTTP Components* (on page 24)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Post() (see *Post() (method)* on page 205)
- type (see *type (property)* on page 288)
- UserAgent (see *UserAgent (property)* on page 291)
- value (see *value (property)* on page 294)
- wlClear() (see *wlClear() (method)* on page 301)

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlLocals (see *wlLocals (object)* on page 319)

host (property)

Property of Object

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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`"

hostname (property)

Property of Object

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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`"

href (property)

Property of Object

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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

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]
```

HttpCacheScope (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Defines when the Http cache will be cleared. Possible values are:

- **None** – Defines that all Virtual Clients simulate a browser with no available cache.

- **SingleCommand** – Defines that cache be cleared after each request.
- **SingleCommandIfModified** – Defines that WebLOAD will check for a newer version of the cached item with every request. Whenever the engine has a request for a cached resource, the engine sends the request with an "if-modified-since" header. If the server responds with a 200 status, the engine will refresh the cache.
- **SingleRound** – Defines that cache be cleared after each script execution round. This is the default value for the HttpCacheScope property.
- **WholeRun** – Defines that the cache will never be cleared. Each client maintains its own cache.
- **WholeRunIfModified** – Defines that WebLOAD will check for a newer version of the cached item after each round. Whenever the engine has a request for a cached resource, the engine sends the request with an "if-modified-since" header. If the server responds with a 200 status, the engine will refresh the cache.

Example

```
wlGlobals.HttpCacheScope = "SingleCommand"
```

GUI mode

In the WebLOAD Recorder, select one of the cache scope options in the Browser Cache tab of the **Default/Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Note: The default value for the cache scope is **SingleRound**.

See also

- HttpCacheCachedTypes (see *HttpCacheCachedTypes (property)* on page 144)

HttpCacheCachedTypes (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Specifies the type of content to include in the HTTP cache: None, HTML, JS, CSS, XML, Applet, Image, Dynamic (a URL with a query string). The default value is **JS**, **CSS**, **XML**, **Applet**, **Image**.

Example

```
wlGlobals.HttpCacheCachedTypes = "Image,CSS"
```

GUI mode

For wlGlobals.HttpCacheCachedTypes, you can also set the Cache Content Filter from WebLOAD Recorder or Console.

In WebLOAD Recorder, in the **Browser Cache** tab of the **Default** or **Current Options** dialog box, select either the **Default** or **User Filter** in the Cache Content Filter area. If you select **User Filter**, check the relevant filters.

In WebLOAD Console, in the **Browser Cache** tab of the **Default** or **Current Options** dialog box or the **Script Options** dialog box, select either the **Default** or **User Filter** in the Cache Content Filter area. If you select **User Filter**, check the relevant filters.

See also

- HttpCacheScope (see *HttpCacheScope (property)* on page 143)

httpEquiv (property)

Property of Object

- wlMetas (see *wlMetas (object)* on page 320)

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 (see *content (property)* on page 56)
- Name (see *Name (property)* on page 174)
- Url (see *Url (property)* on page 289)

HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

- wlLocals (see *wlLocals (object)* on page 319)

Description

Identifies the proxy server that the script uses for HTTP SSL access when `UseSameProxyforSSL` is set to false. The user name and password are for SSL proxy servers that require user authorization. These properties are used when you are working with a separate SSL proxy.

Note: This property can only be inserted manually.

Syntax

```
wlGlobals.httpsProxyProperty = "TextString"
```

Example

```
wlGlobals.httpsProxy = "proxy.ABCDEF.com:8080"
wlGlobals.httpsProxyUserName = "Bill"
wlGlobals.httpsProxyPassWord = "Classified"
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*
- Proxy, ProxyUserName, ProxyPassWord (see *Proxy, ProxyUserName, ProxyPassWord (properties)* on page 210)
- ProxyNTUserName, ProxyNTPassWord (see *ProxyNTUserName, ProxyNTPassWord (properties)* on page 212)
- HttpsProxyNTUserName, HttpsProxyNTPassWord (see *HttpsProxyNTUserName, HttpsProxyNTPassWord (properties)* on page 146)
- UseSameProxyForSSL (see *UseSameProxyForSSL (property)* on page 292)

HttpsProxyNTUserName, HttpsProxyNTPassWord (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Provides user authorization to the proxy server that the script uses for HTTP SSL access on Windows servers when `UseSameProxyforSSL` is set to `false`.

Syntax

```
wlGlobals.httpsProxyNTProperty = "TextString"
```

Example

```
wlGlobals.httpsProxyNTUserName = "Bill"
wlGlobals.httpsProxyNTPassWord = "Classified"
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*
- HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (see *HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (properties)* on page 146)
- Proxy, ProxyUserName, ProxyPassWord (see *Proxy, ProxyUserName, ProxyPassWord (properties)* on page 210)
- ProxyNTUserName, ProxyNTPassWord (see *ProxyNTUserName, ProxyNTPassWord (properties)* on page 212)
- UseSameProxyForSSL (see *UseSameProxyForSSL (property)* on page 292)

id (property)

Property of Objects

- element (see *element (object)* on page 80)
- form (see *form (object)* on page 95)
- frames (see *frames (object)* on page 99)
- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)
- script (see *script (object)* on page 228)
- *Select* (on page 230)
- wlTables (see *wlTables (object)* on page 333)
- wlXmls (see *wlXmls (object)* on page 340)

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 `wlTables` object found with that identifier.

wlXmls 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.wlXmls[0]
```

-Or-

```
MyBookstore = document.wlXmls.myXmlDoc
```

-Or-

```
MyBookstore = document.wlXmls["myXmlDoc"]
```

If duplicate identifiers are found, the `id` property will refer to the first XML object found with that identifier.

See also

- cell (see *cell (object)* on page 44) (`wlTables` and `row` property)
- cellIndex (see *cellIndex (property)* on page 46) (`cell` property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (`wlTables` property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)

- id (see *id (property)* on page 146) (`wlTables` property)
- InnerHTML (see *InnerHTML (property)* on page 154) (`cell` property)
- InnerText (see *InnerText (property)* on page 156) (`cell` property)
- load() (see *load() (method)* on page 163)
- loadXML() (see *loadXML() (method)* on page 167)
- *load() and loadXML() Method Comparison* (on page 164)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (`wlTables` property)
- rowIndex (see *rowIndex (property)* on page 224) (`row` property)
- src (see *src (property)* on page 252)
- tagName (see *tagName (property)* on page 279) (`cell` property)
- *Working with HTTP Protocol* in the *WebLOAD Scripting Guide*
- XMLDocument (see *XMLDocument (property)* on page 345)

Image (object)

Property of Objects

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

Description

Each `Image` object represents one of the images or video clips embedded in a document (HTML <IMG> element). `Image` objects are accessed through `Images` *Collections* (on page 27). (Compare to the element (see *element (object)* on page 80) 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 (see *form (object)* on page 95) object, which stores the parsed data for an entire HTML form.)

`image` objects are grouped together within collections of `images`, accessed directly through the `document` object (`document.images[#]`).

Syntax

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

```
document.images[1].src
```

Properties

- id (see *id (property)* on page 146)
- InnerLink (see *InnerLink (property)* on page 155)
- Name (see *Name (property)* on page 174)
- OuterLink (see *OuterLink (property)* on page 186)
- protocol (see *protocol (property)* on page 210)
- src (see *src (property)* on page 252)
- Url (see *Url (property)* on page 289)

See also

- *Collections* (on page 27)
- form (see *form (object)* on page 95)
- *Select* (on page 230)

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 script execution rounds. Encourages modular programming by enabling easy access to sets of library function files.

Syntax

```
IncludeFile(filename[, WLExecuteScript])
```

Parameters

Parameter Name	Description
`filename`	A string or variable containing the full literal name of the file to be included. WebLOAD assumes that the file is located in the default directory specified in the File Locations tab (User Include Files entry) in the **Tools ➤ Global Options** dialog box in the WebLOAD Console or in the **Tools ➤ Settings** dialog box in the WebLOAD Recorder. For additional information about the included file's location, refer to *Determining the Included File Location* in the *WebLOAD Scripting Guide*. Once the file is found, any functions or variables defined within that file are compiled and included within the calling script when the script 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.

Example

To include the external file `MyFunction.js`, located on the WebLOAD Console during WebLOAD testing, use the following command:

```
function InitAgenda() {
   IncludeFile("MyFunction.js")
}
```

Comment

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

The load engine first looks for the file to be included in the default User Include Files directory. If the file is not there, the file request is handed over to WebLOAD, which searches for the file using the following search path order:

1. If a full path name has been hardcoded into the IncludeFile command, the system searches the specified location. If the file is not found in an explicitly coded directory, the system returns an error code of File Not Found and will not search in any other locations.

Note: It is not recommended to hardcode a full path name, since the script will then not be portable between different systems. This is especially important for networks that use both UNIX and Windows systems.

2. Assuming no hardcoded full path name in the script code, the system looks for the file in the current working directory, the directory from which WebLOAD was originally executed.
3. Finally, if the file is still not found, the system searches for the file sequentially through all the directories listed in the File Locations tab.

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile (see *wlOutputFile (object)* on page 323)
- wlOutputFile() (see *wlOutputFile (object)* on page 323)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

Index (property)

Property of Objects

- frames (see *frames (object)* on page 99)

Description

Sets or retrieves the index number of the parent object. For example, the ordinal position of an option in a list box.

See also

- *Collections* (on page 27)

InfoMessage() (function)

Description

Displays a generally informative (but not necessarily problematic) message in the Log Window.

Syntax

```
InfoMessage(msg)
```

Parameters

Parameter Name	Description
`msg`	A string with an informative message to be sent to the WebLOAD Console.

Comment

If you call `InfoMessage()` in the main script, WebLOAD sends an informative message to the Log window and continues with script 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 script files directly through the WebLOAD Recorder. Message function command lines may also be added directly to the code in a JavaScript Object within a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

See also

- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException (see *wlException (object)* on page 306)
- wlException() (see *wlException() (constructor)* on page 308)

InnerHTML (property)

Property of Objects

- cell (see *cell (object)* on page 44)
- script (see *script (object)* on page 228)
- wlXmls (see *wlXmls (object)* on page 340)

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 `wlXmls` objects, use the lowercase form:

```
…scripts[2].innerHTML
```

Comment

The `InnerHTML` property for `cell` objects is written in uppercase.

See also

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (wlTables property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (wlTables property)
- InnerImage (see *InnerImage (property)* on page 155)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- load() (see *load() (method)* on page 163)
- loadXML() (see *loadXML() (method)* on page 167)
- *load() and loadXML() Method Comparison* (on page 164)
- MatchBy (see *MatchBy (property)* on page 170)

- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (wlTables property)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- src (see *src (property)* on page 252)
- tagName (see *tagName (property)* on page 279) (cell property)
- wlTables (see *wlTables (object)* on page 333)
- XMLDocument (see *XMLDocument (property)* on page 345)

InnerImage (property)

Property of Object

- element (see *element (object)* on page 80)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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.

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- InnerText (see *InnerText (property)* on page 156)
- src (see *src (property)* on page 252)

InnerLink (property)

Property of Objects

- Image (see *Image (object)* on page 149)

Description

Represents the inner link field for the parent `image` object.

See also

- *Collections* (on page 27)
- form (see *form (object)* on page 95)
- *Select* (on page 230)

InnerText (property)

Property of Object

- cell (see *cell (object)* on page 44)
- element (see *element (object)* on page 80)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

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.

See also

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (wlTables property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- element (see *element (object)* on page 80)
- id (see *id (property)* on page 146) (wlTables and wlXmls property)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell and wlXmls property)
- InnerImage (see *InnerImage (property)* on page 155)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)
- MatchBy (see *MatchBy (property)* on page 170)

- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (wlTables property)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- src (see *src (property)* on page 252)
- tagName (see *tagName (property)* on page 279) (cell property)
- wlTables (see *wlTables (object)* on page 333)

JVMType (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The JVMType property indicates the JVM to be used in the Load Generator. The value of this property is defined using the WebLOAD Console or WebLOAD Recorder and overrides the JVM definition in `webload.ini`.

The value (string) of this property is the key for `WLJVMs.xml`. This file (located on every WebLOAD Machine in the `<WebLOAD Installation Directory>\extensions\JVMs` directory) contains the following parameters for each JVM:

- Type (the value from the flag)
- Path (should be machine-agnostic)
- Options

When Type is "Default", the RadView default (installed) JVM will be used. The default JVM's path is defined in `webload.ini`, as it depends on the WebLOAD installation path.

Note: The classpath definitions are defined in `webload.ini`.

GUI mode

In WebLOAD Console, select a JVM from the **Select run-time JVM to be used** drop-down list in the Java Options tab of the **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, select a JVM from the **Select run-time JVM to be used** drop-down list in the Java Options tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

KDCServer (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Specifies the address of the Key Distribution Center (KDC) server if you are using the Kerberos authentication method.

Note: The `KDCServer` property is only relevant for playback.

Syntax

```
KDCServer(ServerName)
```

Parameters

Parameter Name	Description
`ServerName`	The name of the KDC server if you are using the Kerberos authentication method.

Example

```
wlGlobals.KDCServer = "qa4"
```

GUI mode

To specify the name of the KDC server if you are using the Kerberos authentication method:

- In WebLOAD Console, select the Kerberos radio button and enter the address of the KDC server in the Kerberos Server field in the Authentication tab of the **Default**, **Current Session**, or **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.
- In WebLOAD Recorder, select the Kerberos radio button and enter the address of the KDC server in the Kerberos Server field in the Authentication tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

Only the server name should be specified in `KDCServer`. For example, specify "qa4" rather than "qa4.radview.co.il".

See also

- AuthType (see *AuthType (property)* on page 40)

KeepAlive (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enable WebLOAD to keep an HTTP connection alive between successive accesses in the same round of the main script. The possible values are:

- **false** – Do not keep an HTTP connection alive.
- **true** – 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.

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 WebLOAD Console. Enable maintaining connections for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)
- CloseConnection() (see *CloseConnection() (method)* on page 53)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*

- *Working with HTTP Protocol* in the *WebLOAD Scripting Guide*

KeepRedirectionHeaders (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

Used to indicate whether to get the location headers of all redirections. The default value of `KeepRedirectionHeaders` is false.

Example

```
wlGlobals.KeepRedirectionHeaders = true
```

Comment

This property is useful for the following scenario, which occurs in correlation. During a redirection, in the middle of one of the URLs, there is a parameter in the Location header that is needed for the next Get. Since only the headers of the last Get in a series of redirections are stored in `document.wlHeaders`, the `KeepRedirectionHeaders` property, when set to `true`, enables all the headers in a series of redirections to be saved. The value can be extracted from `document.wlHeaders` after the navigation is complete.

See also

- SaveHeaders (see *SaveHeaders (property)* on page 225)

key (property)

Property of Objects

- Header (see *Header (property)* on page 140)
- wlHeaders (see *wlHeaders (object)* on page 314)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

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

- value (see *value (property)* on page 294)

language (property)

Property of Object

- script (see *script (object)* on page 228)

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.

link (object)

Property of Objects

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.

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 (see *Collections* on page 27) 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

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>
```

Example

```
document.links[1].protocol
```

Properties

- hash (see *hash (property)* on page 139)
- host (see *host (property)* on page 142)
- hostname (see *hostname (property)* on page 142)
- href (see *href (property)* on page 143)
- id (see *id (property)* on page 146)
- InnerImage (see *InnerImage (property)* on page 155)
- InnerText (see *InnerText (property)* on page 156)
- Name (see *Name (property)* on page 174)
- pathname (see *pathname (property)* on page 204)

- port (see *port (property)* on page 204)
- protocol (see *protocol (property)* on page 210)
- search (see *search (property)* on page 229)
- target (see *target (property)* on page 280)
- title (see *title (property)* on page 284)
- Url (see *Url (property)* on page 289)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

See also

- *Collections* (on page 27)
- document (see *document (object)* on page 78)

load() (method)

Method of Objects

XML DOM objects on a Web page are accessed through collections of `wlXmls` objects. The `load()` function is a method of the following object:

- wlXmls (see *wlXmls (object)* on page 340)

Description

Call `load(URL)` to download XML documents from a website 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 performs 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 script 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

Parameter Name	Description
URLString	String parameter with the URL or filename where the XML document may be found.

Example

```
myXMLDoc = document.wlXmls[0]
myXMLdoc.load("http://server/xmls/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.

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- loadXML() (see *loadXML() (method)* on page 167)
- *load() and loadXML() Method Comparison* (on page 164)
- src (see *src (property)* on page 252)
- XMLDocument (see *XMLDocument (property)* on page 345)

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 5: load() and loadxml() Comparison

	Advantages	Disadvantages
loadXML()	Parameters that the user has defined through WebLOAD for the testing session will be applied to this transaction.	The method fails if the DTD section of the XML document string includes any external references.

	Advantages	Disadvantages
load()	The user may load XML files that include external references in the DTD section.	Parameters that the user has defined through WebLOAD for the testing session will not be applied to this transaction. WebLOAD does not record the HTTP Get operation. The transaction results are not included in the session statistics report. Using this method may adversely affect the test session results.

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.wlXmls[0]
SetTimer("GetXMLTime")
myXMLdoc.load("http://server/xmls/file.xml")
SendTimer("GetXMLTime")
```

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- load() (see *load() (method)* on page 163)
- loadXML() (see *loadXML() (method)* on page 167)
- src (see *src (property)* on page 252)
- wlXmls (see *wlXmls (object)* on page 340)
- XMLDocument (see *XMLDocument (property)* on page 345)

LoadGeneratorThreads (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

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 script. In that case, if you configure a Probing Client to run the script, WebLOAD uses the value of `ProbingClientThreads` and ignores `LoadGeneratorThreads` (vice versa if you configure a Load Generator to run the script). For example, you might write:

```
function InitAgenda() {
  //Do not use extra threads if a
  // Probing Client runs the script
  wlGlobals.ProbingClientThreads = "Single"
  //Use extra threads if a
  // Load Generator runs the script
  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 WebLOAD Console. Enable multi-threading for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default** or **Current Session Options** dialog box and setting the number of threads you prefer.

See also

- *HTTP Components* (on page 24)
- ProbingClientThreads (see *ProbingClientThreads (property)* on page 208)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*

loadXML() (method)

Method of Object

XML DOM objects on a Web page are accessed through collections of `wlXmls` objects. The `loadXML()` function is a method of the following objects:

- wlXmls (see *wlXmls (object)* on page 340)

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.

Note: 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

Parameter Name	Description
`XMLDocStr`	String parameter that contains a literal XML document in string format.

Example

```
//create a new XML document object
NewXMLObj = new WLXmlDocument()
wlHttp.SaveSource = true
wlHttp.Get("http://www.server.com/xmls/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.

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- load() (see *load() (method)* on page 163)
- *load() and loadXML() Method Comparison* (on page 164)
- src (see *src (property)* on page 252)
- XMLDocument (see *XMLDocument (property)* on page 345)

location (object)

Property of Objects

- document (see *document (object)* on page 78)

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* in the *WebLOAD Scripting 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>
```

Properties

Note: The properties of `location` are identical to those of `link`. The only exception is that `location` has no `target` property. Also, the `location` object is not part of any collection. The `location` properties are listed below for reference.

- hash (see *hash (property)* on page 139)
- host (see *host (property)* on page 142)
- hostname (see *hostname (property)* on page 142)
- href (see *href (property)* on page 143)
- id (see *id (property)* on page 146)
- InnerText (see *InnerText (property)* on page 156)
- Name (see *Name (property)* on page 174)
- pathname (see *pathname (property)* on page 204)
- port (see *port (property)* on page 204)
- protocol (see *protocol (property)* on page 210)
- search (see *search (property)* on page 229)
- title (see *title (property)* on page 284)
- Url (see *Url (property)* on page 289)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

Comment

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

- link (see *link (object)* on page 162)

MaxLength (property)

Property of Object

- element (see *element (object)* on page 80)

Description

The maximum number of characters the user can enter into a Text or Password element.

MaxPageTime (function)

Description

Verifies the PageTime of the service response. If the PageTime (time to download the page) exceeds the specified maximum value, the verification fails.

Syntax

```
wlVerification.MaxPageTime(timeLimit, severity)
```

Parameters

Parameter Name	Description
timeLimit	The maximum amount of time to download the page in seconds.
`severity`	Possible values of this parameter are: • `WLSuccess`. The transaction terminated successfully. • `WLMinorError`. This specific transaction failed, but the test session may continue as usual. The script 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 script 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 script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- Severity (see *Severity (property)* on page 247)

- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

method (property)

Property of Object

- form (see *form (object)* on page 95)

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`".

MultiIPSupport (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

WebLOAD enables use of all available IP addresses. This allows testers to simulate clients with different IP addresses using only one Load Generator. You must first generate additional IP addresses on your machine to use when testing and then set the `MultiIPSupport` property to true to enable multiple IP support. For more information about generating additional IP addresses, see *Generating IP Addresses in the script* in the *WebLOAD Scripting Guide*.

Note: Setting the `MultiIPSupport` property to true without generating additional IP addresses on your machine will not enable multiple IP support.

The possible values of `wlGlobals.MultiIPSupport` are:

- **false** – Use only one IP address. (default)
- **true** – Use all available IP addresses.

When connecting Load Generators through a modem, `MultiIPSupport` should be set to `false`.

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 `MultiIPSupport` property.

GUI mode

In WebLOAD Console, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

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 `true`, make sure that all of the Applications Being Tested to which the script refers are accessible through all the network interface cards.

Use the GetIPAddress() (see *GetIPAddress() (method)* on page 122) method to check the identity of the current IP address.

See also

- *HTTP Components* (on page 24)
- GetIPAddress() (see *GetIPAddress() (method)* on page 122)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*
- MultiIPSupportType() (see *MultiIPSupportType (property)* on page 172)

MultiIPSupportType (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The MultiIPSupportType property works with the `wlGlobals.MultiIPSupport` property, and supports the following values:

- **PerClient** (default) – Preserves the current behavior. This means that there are different IPs per client but the same IP is used for all rounds.
- **PerRound** – Supports the use of a different IP from the pool per client, per round, until the pool is exhausted, after which it returns to the beginning.

This property is only referenced when `wlGlobals.MultiIPSupport` is true.

Note: To support multiple IP addresses, you must generate additional IP addresses on your machine and then set the `MultiIPSupport` property to true. For more information about generating additional IP addresses, see *Generating IP Addresses in the script* in the *WebLOAD Scripting Guide*.

GUI mode

In WebLOAD Console, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

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 true, make sure that all of the Systems under Test (SUT) to which the script refers are accessible through all the network interface cards.

Use GetIPAddress() (see *GetIPAddress() (method)* on page 122) to check the identity of the current IP address.

See also

- *HTTP Components* (on page 24)
- GetIPAddress() (see *GetIPAddress() (method)* on page 122)
- MultiIPSupport() (see *MultiIPSupport (property)* on page 171)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*

MultiIPSupportProtocol (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The MultiIPSupportProtocol property works with the `wlGlobals.MultiIPSupport` property, and supports the following values:

- **All** (default) – Support both the IPv4 and IPv6 protocols.
- **IPv4Only** – Support only the IPv4 IP protocol.
- **IPv6Only** – Support only the IPv6 IP protocol.

This property is only referenced when `wlGlobals.MultiIPSupport` is true.

GUI mode

In WebLOAD Console, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Recorder, check or uncheck **Multi IP Support** in the HTTP Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)
- GetIPAddress() (see *GetIPAddress() (method)* on page 122)
- MultiIPSupport() (see *MultiIPSupport (property)* on page 171)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*

Name (property)

Property of Objects

- element (see *element (object)* on page 80)
- form (see *form (object)* on page 95)
- frames (see *frames (object)* on page 99)
- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)
- *Select* (on page 230)

- wlMetas (see *wlMetas (object)* on page 320)

Description

Sets or retrieves the identification string of the parent object.

Note: 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). The 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
```

See also

- *Collections* (on page 27)
- content (see *content (property)* on page 56)
- httpEquiv (see *httpEquiv (property)* on page 144)
- Url (see *Url (property)* on page 289)

NTUserName, NTPassWord (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The user name and password that the script uses for Windows NT Challenge response authentication (NT Challenge Response).

Comments

A user is only authenticated once during a round with a set of credentials. Each subsequent request will use these credentials regardless of what is contained in the script. If the value of these credentials are changed after authentication, they will only be used during the next round, not during the current round.

For example, if you are trying to send a request to a URL with a group of users (user1, user2, and user3), but user1 has already been authenticated, the login is always performed for user1 until the round is complete.

GUI mode

By default, WebLOAD 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 WebLOAD Console. Enter user authentication information through the Authentication tab of the **Default** or **Current Options** dialog box, accessed from the **Tools** tab of the ribbon.

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 script is running.

See also

- *HTTP Components* (on page 24)
- *Rules of Scope for Local and Global Variables* in the *WebLOAD Scripting Guide*
- *Working with HTTP Protocol* in the *WebLOAD Scripting Guide*

Num() (method)

Method of Object

- wlRand (see *wlRand (object)* on page 326)

Description

Return a random integer.

Syntax

```
wlRand.Num([seed])
```

Parameters

Parameter Name	Description
`[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

- Range() (see *Range() (method)* on page 215)
- Seed() (see *Seed() (method)* on page 229)
- Select() (see *Select() (method)* on page 230)

onDataReceived (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

Description

Define a callback function to be called every time more data is received for the request. This is useful for working with asynchronous requests (when `wlHttp.Async=true`) that need to be inspected before they are completed, for example in an HTTP streaming push scenario.

Use the callback function to handle the asynchronous request, for example – validate the response and make a further request.

The callback function argument is a *limited* 'document' object. The document object contains only the following properties:

- wlSource (see *wlSource (property)* on page 330)
- wlStatusNumber (see *wlStatusNumber (property)* on page 331)
- wlStatusLine (see *wlStatusLine (property)* on page 331)

Note: The callback is expected to run in a timely manner, because it blocks the execution of other callback functions. Specifically, try to:
Avoid making synchronous HTTP requests – use asynchronous ones instead.
Avoid using Sleep() inside a callback function – instead, use *setTimeout() (function)* to execute code after a certain period of time.

Note: The `onDataRecived` callback is called many times – each time more data is received. If you only need to inspect the complete response, use the *onDocumentComplete (property)* callback instead.

Example

```
wlHttp.Async = true;
wlHttp.onDataReceived = function(document) {
{
  if (document.wlStatusNumber==200)
    {
  InfoMessage( "Got response so far: " + document.wlSource );
    }
}
wlHttp.Get("http:///.....")
```

See also

- *HTTP Components* (on page 24)
- The *Using Asynchronous Requests* chapter in the *WebLOAD Scripting Guide*
- *wlSource (property)* (on page 330)
- *wlStatusNumber (property)* (on page 331)
- *wlStatusLine (property)* (on page 331)

- *Async (property)* (on page 41)
- *onDocumentComplete (property)* (on page 179)

onDocumentComplete (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

Description

Define a callback function to be called after the request has been completed. Useful in asynchronous requests (when `wlHttp.Async=true`)

Use the callback function to handle the asynchronous request, for example – validate the response and make a further request.

The callback function argument is the 'document' object, containing the response data, headers, status, etc.

Note: The callback is expected to run in a timely manner, because it blocks the execution of other callback functions. Specifically, try to:
Avoid making synchronous HTTP requests – use asynchronous ones instead.
Avoid using Sleep() inside a callback function – instead, use *setTimeout() (function)* to execute code after a certain period of time.

Note: The `onDocumentComplete` callback is called only once – when the request is fully completed. To handle partial responses, use the *onDataReceived (property)* callback.

Example

```
wlHttp.Async = true;
wlHttp.onDocumentComplete = function(document) {
   InfoMessage("Response " + document.wlSource);
}
wlHttp.Get("http://get-asynch-data");
//the script will continue to here immediately, not waiting for the
request to complete. It will run the onDocumentComplete function when
the request is finished.
```

See also

- *HTTP Components* (on page 24)
- The *Using Asynchronous Requests* chapter in the *WebLOAD Scripting Guide*
- *document (object)* (on page 78)
- *Async (property)* (on page 41)

- *onDataReceived (property)* (on page 177)

Open() (method)

Method of Object

- wlInputFile (see *wlInputFile (object)* on page 317)

Description

Opens the input file specified in the `wlInputFile` object. This should be done in the `InitClient` section of your script.

Syntax

```
function InitAgenda()
{
  …
  fileID = CopyFile(<full path>)
  …
}
function InitClient()
{
  …
  MyFileObj = new wlInputFile(fileID)
  MyFileObj.Open([AccessMethod], [ShareMethod], [UsageMethod],
  [EndOfFileBehavior], [HeaderLines], ['Delimiter'])
  …
}
```

Parameters

Parameter Name	Description
AccessMethod	An optional parameter that defines the method for reading the next value/row from the file. All values are enumerated numeric values. Possible values are: • WLFileSequential. Every client gets the next value/row from the file, where there might be multiple access to the same line by different Load Generator machines. This is the default value. • WLFileSequentialUnique. Gets the next unique value/row from the file. Preferably, the unique value is the next available value in sequential order. If another VC is using this value/row, the VC is not able to access this value/row and will get the next available value/row. It is recommended to have more values/rows in the file than the number of clients to avoid delays. • WLFileRandom. Gets a random value/row from the file. There might be multiple access to the same line by different Load Generator machines. • WLFileRandomUnique. Gets a unique, unused value/row randomly from the file. It is recommended to have more values/rows in the file than the number of clients to avoid delays.
ShareMethod	An optional parameter indicating how the file is shared among scripts. All values are enumerated numeric values. Possible values are: • **WLFileNotShared**. The file can be read only by the current script, and each Load Generator machine manages a copy of the file for its VCs independently. If there are multiple Load Generator processes on a single machine, then the processes share the file. This is the default value. • **WLFileLGShared**. The file can be read only by the current script, and all Load Generators on any Load Generator machine share the same copy of the file, which is synchronized between them. • **WLFileAgendaShared**. The file can be shared by more than one script. The unique identifier of the file is its path. The file can be shared by different scripts, but a copy of the file is managed separately for each Load Generator machine. If you are using the script–Shared share method, all the scripts sharing the file should use the WLFileSequentialUnique access method. • **WLFileAgendaLGShared**. A single file is shared among Load Generators and among scripts.

Parameter Name	Description
`UsageMethod`	An optional parameter that defines when to release the value/row back to the 'pool' so that it can be read again from the file. This parameter is only relevant for the WLFileSequentialUnique and WLFileRandomUnique access methods. All values are enumerated numeric values. Possible values are: • **WLFilePerRound**. The script reads a new value/row from the file once every round. The value/row is released at the end of the round. This is the default value. • **WLFileOncePerClient**. The script reads a new value/row from the file once at the beginning of the test (in InitClient). The value/row is released at TerminateClient. • **WLFileOncePerSession**. The script reads a new value/row from the file once, at the beginning of the session (in InitClient). The value/row is released at the end of the session (in TerminateAgenda). • **WLFileAnytime**. The script can read a new value/row from the file at any time during a round. It can read a new value/row more than once during a round. The values are released at the end of the round. This enables more than one value/row to be used concurrently and uniquely.
`EndOfFileBehavior`	An optional parameter that defines how WebLOAD behaves when it reaches the end of the file. All values are enumerated numeric values.  **Note:** If you have defined the `AccessMethod` as WLFileSequential or WLFileSequentialUnique, the `EndOfFileBehavior` parameter is mandatory. Possible values are: • **WLFileStartOver**. Start from the beginning of the file. This is the default value. • **WLFileKeepLast**. Keep the last value. • **WLFileAbortVC**. Abort the specific VC that tried to read past the end of the file. An error message is written to the log file. • **WLFileAbortTest**. Abort the entire test when a VC tries to read past the end of the file. An error message is written to the log file.

Parameter Name	Description
HeaderLines	An optional parameter that defines the number of header lines the file contains. All values are enumerated numeric values. Possible values are: • 0. The file does not contain any header lines. This is the default value. • <X>. Where <X> is any number above zero. The file contains <X> header lines at its beginning. The values contained in these header lines are not used as parameters but as variable names in the JavaScript code.
Delimiter	(Optional) The delimiter being used in the file. The default value is a comma.

Example

```
function InitAgenda()
{
  InFile1 = CopyFile("C:\\temp\input.txt")
}
Function InitClient()
{
  myFileObj = new wlInputFile(InFile1)
  myFileObj.open(WLFileSequentialUnique, WLFileAgendaShared)
}
/*** WLIDE …. ***/
strLine = myFileObj.getLine(",")
```

See also

- CopyFile() (see *CopyFile() (function)* on page 61)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- wlInputFile() (see *wlInputFile() (constructor)* on page 318)

Open() (function)

Method of Object

- wlOutputFile (see *wlOutputFile (object)* on page 323)

Description

Opens the output file, specified in the wlOutputFile object. By default, the file is opened for sequential access, enabling the parameters in the file to be read sequentially. This is unique across the master and slave processes of a single Load Generator/script combination. The master assigns the next line of the file that will be

read sequentially for each slave. When all the information in the file is read (see *GetLine() (function)* on page 123), it is returned to the beginning of the file.

Alternatively, to open the input file and read its contents in random order, you must include `Open(filename, wlRandom)` in the script's `InitAgenda()` function.

Note: The last line of the file should not include a carriage return.

Syntax

For sequential access:

```
MyFileObj = new wlOutputFile(filename)
…
MyFileObj.Open()
```

For random access:

```
Open(filename, wlRandom)
```

Parameters

Parameter Name	Description
`filename`	The name of the file to be opened.
`wlRandom`	A flag indicating that the file should be opened in random access mode.

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Reset() (see *Reset() (method)* on page 220)
- *Using the Form Data Wizard* in the *WebLOAD Scripting Guide*
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile() (see *wlOutputFile() (constructor)* on page 324)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

option (object)

Property of Object

`Option` objects are grouped into collections of `options` that are themselves properties of the following:

- element (see *element (object)* on page 80)
- *Select* (on page 230)

Description

A collection of the nested <OPTION> objects only found within elements of type SELECT, that is, `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

Options only exist if the type of the parent element is <SELECT>, that is, `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 script:

```
function InitAgenda()
{
  wlGlobals.Proxy = "webproxy.xyz.com:8080"
```

```
                        // Through proxy
  wlGlobals.SaveSource = true
  wlGlobals.ParseForms = true
  wlGlobals.ParseTables = true
}
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 (see *defaultselected (property)* on page 72)
- selected (see *selected (property)* on page 235)
- value (see *value (property)* on page 294)

Options() (method)

Method of Objects

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

- wlHttp (see *wlHttp (object)* on page 316)

Description

Perform an HTTP or HTTPS Options command.

Syntax

```
Options([URL])
```

Parameters

Parameter Name	Description
`[URL]`	An optional parameter identifying the document URL. You may optionally specify the URL as a parameter of the method. `Options()` connects to the first URL that has been specified from the following list, in the order specified: • A `Url` parameter specified in the method call. • The `Url` property of the `wlHttp` object. • The local default `wlLocals.Url`. • The global default `wlGlobals.Url`.

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Delete() (see *Delete() (HTTP method)*) on page 74
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Head() (see *Head() (method)* on page 139)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- Put() (see *Put() (method)* onpage 213)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

OuterLink (property)

Property of Objects

- Image (see *Image (object)* on page 149)

Description

Represents the outer link field for the parent `image` object.

See also

- *Collections* (on page 27)
- form (see *form (object)* on page 95)
- *Select* (on page 230)

Outfile (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

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.

The `Outfile` property is used to implement the Log Report.

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 *script output* messages, use the wlOutputFile. (see *wlOutputFile (object)* on page 323)

See also

- wlOutputFile (see *wlOutputFile (object)* on page 323)

PageContentLength (property)

Property of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

`PageContentLength` is used to retrieve the size in bytes of the content object in the GET/POST request. The content object may only be HTML, ASP, or JPG.

Syntax

```
wlVerification.PageContentLength
```

Example

```
wlHttp.Get("http://www.google.com/")
InfoMessage("page size" + wlVerification.PageContentLength)
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageTime (see *PageTime (property)* on page 190)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

PageTime (property)

Property of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

`PageTime` is used to retrieve the page time of the last GET. That is, the total time taken to retrieve the page.

Syntax

```
wlVerification.PageTime
```

Example

```
wlHttp.Get("http://www.google.com/")
InfoMessage("page time" + wlVerification.PageTime)
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

Parse (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing on an HTML page.

The Parse property can be set to one of the following values:

- **always** (default) – Each page is parsed and the DOM is created every time the page is visited.
- **OnceOnly** – The page is parsed and the DOM is created only the first time the page is visited. The same data is then reused on future visits.

- **no** – The DOM is not created and no object can be retrieved.

Note: If you want the page to be parsed and the DOM created the first time the page is visited and then reuse this data, set the ParseOnce property to true. For information about the ParseOnce property, see *ParseOnce (property)* on page 198.

Note: This property can only be inserted manually.

Syntax

```
wlGlobals.Parse = "Always"
```

See also

- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseApplets (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables/disables parsing of Java applets on an HTML page. The ParseApplets property can be set to one of the following values:

- **true** (default) – Enables parsing of Java applets.
- **false** – Disables parsing of Java applets.

Note: This property can only be inserted manually.

Note: If GetApplets is true, ParseApplets will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseApplets = false
```

See also

- Parse (see *Parse (property))* on page 190)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseCss (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of cascading style sheets on an HTML page. The ParseApplets property can be set to one of the following values:

- **true** (default) – Enables parsing of cascading style sheets.
- **false** – Disables parsing of cascading style sheets.

Note: This property can only be inserted manually.

Note: If GetCss is true, ParseCss will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseCss = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseEmbeds (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of embedded objects on an HTML page. The ParseEmbeds property can be set to one of the following values:

- **true** (default) – Enables parsing of embedded objects.
- **false** – Disables parsing of embedded objects.

Note: This property can only be inserted manually.

Note: If GetEmbeds is true, ParseEmbeds will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseEmbeds = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseForms (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of forms on an HTML page. The ParseForms property can be set to one of the following values:

- **true** (default) – Enables parsing of forms.
- **false** – Disables parsing of forms.

Note: This property can only be inserted manually.

Note: If GetForms is true, ParseForms will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseForms = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseImages (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of images on an HTML page. The ParseImages property can be set to one of the following values:

- **true** (default) – Enables parsing of images.
- **false** – Disables parsing of images.

Note: This property can only be inserted manually.

Note: If GetImages is true, ParseImages will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseImages = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseLinks (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of links and areas on an HTML page. The ParseLinks property can be set to one of the following values:

- **true** (default) – Enables parsing of links.
- **false** – Disables parsing of links.

Note: This property can only be inserted manually.

Note: If GetLinks is true, ParseLinks will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseLinks = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseMetas (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of metas on an HTML page. The ParseMetas property can be set to one of the following values:

- **true** (default) – Enables parsing of metas.
- **false** – Disables parsing of metas.

Note: This property can only be inserted manually.

Note: If GetMetas is true, ParseMetas will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseMetas = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseOnce (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

When set to `true`, the webpage is parsed and the DOM is created only the first time the page is visited. The same data is reused on future visits. The ParseOnce property is set when you call `SetClientType("Thin")`. By default, the ParseOnce property is set to `true`.

Note: This property can only be inserted manually.

Example

```
wlGlobals.ParseOnce = true
```

See also

- Parse (see *Parse (property)*on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)

- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)
- SetClientType (see *SetClientType (function)* on page 242)

ParseOthers (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing on an HTML page for all objects not covered by specific parsing properties. The ParseOthers property can be set to one of the following values:

- **true** (default) – Enables parsing of other objects.
- **false** – Disables parsing of other objects.

Note: This property can only be inserted manually.

Example

```
wlGlobals.ParseOthers = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)

- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseScripts (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of JavaScript scripts on an HTML page. The ParseScripts property can be set to one of the following values:

- **true** (default) – Enables parsing of JavaScript scripts.
- **false** – Disables parsing of JavaScript scripts.

Note: This property can only be inserted manually.

Note: If GetScripts is true, ParseScripts will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseScripts = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)

- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseTables (see *ParseTables (property)* on page 201)
- ParseXML (see *ParseXML (property)* on page 202)

ParseTables (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of tables on an HTML page. The ParseTables property can be set to one of the following values:

- **true** (default) – Enables parsing of tables.
- **false** – Disables parsing of tables.

Note: This property can only be inserted manually.

Note: If GetTables is true, ParseTables will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseTables = true
```

See also

- Parse (see *Parse (property)* on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)

- ParseScripts (see *ParseScripts (property)* on page 200)
- ParseXML (see *ParseXML (property)* on page 202)

ParseXML (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enables parsing of XML on an HTML page. The ParseXML property can be set to one of the following values:

- **true** (default) – Enables parsing of XML.
- **false** – Disables parsing of XML.

Note: This property can only be inserted manually.

Note: If GetXML is true, ParseXML will automatically be assumed to be true, even if it is set to false.

Example

```
wlGlobals.ParseXML = true
```

See also

- Parse (see *Parse (property)*on page 190)
- ParseApplets (see *ParseApplets (property)* on page 191)
- ParseCss (see *ParseCss (property)* on page 192)
- ParseEmbeds (see *ParseEmbeds (property)* on page 193)
- ParseForms (see *ParseForms (property)* on page 194)
- ParseImages (see *ParseImages (property)* on page 195)
- ParseLinks (see *ParseLinks (property)* on page 196)
- ParseMetas (see *ParseMetas (property)* on page 197)
- ParseOnce (see *ParseOnce (property)* on page 198)
- ParseOthers (see *ParseOthers (property)* on page 199)
- ParseScripts (see *ParseScripts (property)* on page 200)

- ParseTables (see *ParseTables (property)* on page 201)

PassWord (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The password that the script 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.

Comments

A user is only authenticated once during a round with a set of credentials. Each subsequent request will use these credentials regardless of what is contained in the script. If the value of these credentials are changed after authentication, they will only be used during the next round, not during the current round.

For example, if you are trying to send a request to a URL with a group of users (user1, user2, and user3), but user1 has already been authenticated, the login is always performed for user1 until the round is complete.

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 WebLOAD Console by entering user authentication information through the Authentication tab of the **Default** or **Current Options** dialog box, accessed from the **Tools** tab of the ribbon.

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

- *HTTP Components* (on page 24)
- *Working with the HTTP Protocol* in the *WebLOAD Scripting Guide*

pathname (property)

Property of Objects

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

Description

The URI portion of the URL, including the directory path and filename (read-only string).

Example

```
"/products/order.html"
"/search.exe"
```

port (property)

Property of Objects

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

Description

The port of the URL (read-only integer).

Example

```
80
```

Post() (method)

Method of Object

- wlHttp (see *wlHttp (object)* on page 316)

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

Parameter Name	Description
[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: • A `Url` parameter specified in the method call. • The `Url` property of the `wlHttp` object. • The local default `wlLocals.Url`. • 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 the *Data Drilling-WebLOAD transaction reports* section of the *WebLOAD Scripting 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. In the WebLOAD 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 script. The name assigned by WebLOAD is simply the URL used by that Get or Post transaction. If your script includes multiple transactions to the same URL, the information will be collected cumulatively for those transactions.

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")
```

Comment

You may not use the `TransName` parameter by itself. `Post()` expects to receive either *no* parameters, in which case it uses the script'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() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Delete() (see *Delete() (HTTP method)*) on page 74
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Head() (see *Head() (method)* on page 139)
- Options() (see *Options() (method)* on page 186)
- Put() (see *Put() (method)* onpage 213)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

ProbingClientThreads (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

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 script. In that case, if you configure a Probing Client to run the script, WebLOAD uses the value of `ProbingClientThreads` and ignores `LoadGeneratorThreads` (vice versa if you configure a Load Generator to run the script). For example, you might write:

```
function InitAgenda() {
   //Do not use extra threads if a
   // Probing Client runs the script
   wlGlobals.ProbingClientThreads = "Single"
   //Use extra threads if a
   // Load Generator runs the script
   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 WebLOAD Console. Enable multi-threading for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default** or **Current Session Options** dialog box and setting the number of threads you prefer

See also

- *HTTP Components* (on page 24)
- LoadGeneratorThreads (see *LoadGeneratorThreads (property)* on page 165)

protocol (property)

Property of Objects

- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

Description

The HTTP protocol portion of the URL for the parent object (read-only string).

Example

```
"https://"
```

Proxy, ProxyUserName, ProxyPassWord (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Identifies the proxy server that the script 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 WebLOAD Console in one of the following ways:

- Enter user authentication information through the Authentication tab of the **Default** or **Current Options** dialog box, accessed from the **Tools** tab of the ribbon.
- 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.

Syntax

```
wlGlobals.ProxyProperty = "TextString"
```

Example

```
wlGlobals.Proxy = "proxy.ABCDEF.com:8080"
wlGlobals.ProxyUserName = "Bill"
wlGlobals.ProxyPassWord = "Classified"
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*
- HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (see *HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (properties)* on page 145)
- HttpsProxyNTUserName, HttpsProxyNTPassWord (see *HttpsProxyNTUserName, HttpsProxyNTPassWord (properties)* on page 146)
- ProxyNTUserName, ProxyNTPassWord (see *ProxyNTUserName, ProxyNTPassWord (properties)* on page 212)

ProxyExceptions (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

The `ProxyExceptions` property accepts a string based on what the user entered in the Proxy Options tab of the Recording and Script Generation Options dialog box. This string indicates the URLs whose support does not go through the proxy. The format of this string is based on the Internet Explorer format. For more information, see http://www.microsoft.com/technet/prodtechnol/ie/ieak/techinfo/deploy/60/en/corpproxy.mspx?mfr=true.

Example

```
wlGlobals.ProxyExceptions = "*.example.com"
```

GUI mode

WebLOAD takes the current settings of the browser and displays them in the Proxy Options tab of the Recording and Script Generation Options dialog box in WebLOAD Recorder.

In WebLOAD Recorder, click **Recording and Script Generation Options** in the **Tools** tab of the ribbon, and click the Proxy Options tab. Modify the fields, as necessary.

ProxyNTUserName, ProxyNTPassWord (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Provides user authorization to the proxy server that the script uses for HTTP access on Windows servers.

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 WebLOAD Console in one of the following ways:

- Use the Authentication tab of the **Default** or **Current Options** dialog box to enter user authentication information.
- You may also set proxyNT user values using the `wlGlobals` properties. WebLOAD automatically connects via the proxy when a `wlHttp` object connects to an HTTP site.

Syntax

```
wlGlobals.ProxyNTProperty = "TextString"
```

Example

```
wlGlobals.ProxyNTUserName = "Bill"
wlGlobals.ProxyNTPassWord = "Classified"
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*
- HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (see *HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (properties)* on page 145)
- HttpsProxyNTUserName, HttpsProxyNTPassWord (see *HttpsProxyNTUserName, HttpsProxyNTPassWord (properties)* on page 146)
- Proxy, ProxyUserName, ProxyPassWord (see *Proxy, ProxyUserName, ProxyPassWord (properties)* on page 210)

Put() (method)

Method of Object

- wlHttp (see *wlHttp (object)* on page 316)

Description

Perform an HTTP or HTTPS Put command. The method sends the `FormData`, `Data`, or `DataFile` properties in the Put command. In this way, you can submit any type of data to an HTTP server.

Syntax

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

Parameters

Parameter Name	Description
[URL]	An optional parameter identifying the document URL. You may optionally specify the URL as a parameter of the method. Put() connects to first URL that has been specified from the following list: • A `Url` parameter specified in the method call. • The `Url` property of the `wlHttp` object. • The local default `wlLocals.Url`. • The global default `wlGlobals.Url`. The URL must be a server that accepts the submitted data.

`[TransName]`	An optional user-supplied string with the transaction name as it will appear in the Statistics Report, described in the *Data Drilling-WebLOAD transaction reports* section of the *WebLOAD Scripting 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. In the WebLOAD 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, Post and Put 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, Post and Put command in your script. The name assigned by WebLOAD is simply the URL used by that Get, Post or Put transaction. If your script includes multiple transactions to the same URL, the information will be collected cumulatively for those transactions.

Comment

You may not use the `TransName` parameter by itself. `Put()` expects to receive either *no* parameters, in which case it uses the script'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() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Delete() (see *Delete() (HTTP method)*) on page 74
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Head() (see *Head() (method)* on page 139)
- Options() (see *Options() (method)* on page 186)
- Post() (see *Post() (method)* on page 205)

- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

Range() (method)

Method of Object

- wlRand (see *wlRand (object)* on page 326)

Description

Return a random integer between `start` and `end`.

Syntax

```
wlRand.Range(start, end, [seed])
```

Parameters

Parameter Name	Description
`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() (see *Num() (method)* on page 177)
- Seed() (see *Seed() (method)* on page 229)
- Select() (see *Select() (method)* on page 230)

ReceiveTimeout (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

WebLOAD performs read operations in a loop. Each iteration of the loop consists of a wait on the socket until the server is ready, followed by a receive operation, if the read on the socket was successful. This is performed until all the information is read, or until the time spent in the loop exceeds the specified timeout value in the ReceiveTimeout property, or a socket error occurs. If a timeout or socket error occur, WebLOAD then tries to reestablish a connection (see *RequestRetries (property)* on page 220). The default value of the ReceiveTimeout property is 900,000 ms.

Example

```
wlGlobals.ReceiveTimeout = 550000
```

See also

- *HTTP Components* (on page 24)
- RequestRetries (see *RequestRetries (property)* on page 220)

RedirectionLimit (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

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 WebLOAD Console. Check Redirection Enabled and enter a limiting number on the Browser Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

Syntax

You may also assign a redirection limit value using the `wl.RedirectionLimit` property.

```
wlGlobals.RedirectionLimit = IntegerValue
```

Example

```
wlGlobals.RedirectionLimit = 10
```

Referer (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The Referer property is used by the recorder to store the referer header and is a synonym for wlHTTP.Headers[“referer”]. The Referer property is used as shorthand for accessing the referer header in the wlHTTP.Headers collection.

GUI mode

To tell the system whether or not to record the referer header in the Referer property, select or deselect the **Record Referer Header** checkbox in the Script Content tab of the **Recording and Script Generation Options** dialog box, accessed from the **Tools** tab of the ribbon.

Syntax

```
wlHttp.Header[“Referer”] = “http://www.testaddress.com/”.
```

Example

```
wlHttp.Header[“Referer”] = “http://www.easycar.com/”
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*

remove() (method)

Method of Objects

- wlOutputFile (see *wlOutputFile (object)* on page 323)

Description

This method deletes the `wlOutputFile` object and closes the output file.

Syntax

```
wlOutputFile.remove()
```

Example

```
MyFileObj = new wlOutputFile(filename)
…
MyFileObj.remove()
```

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

ReportEvent() (function)

Description

This function enables you to record specific events as they occur. This information is very helpful when analyzing website performance with Data Drilling.

Syntax

```
ReportEvent(EventName[, description])
```

Parameters

Parameter Name	Description
`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.

See also

- CreateDOM() (see *CreateDOM() (function)* on page 63)
- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

ReportLog() (method)

Method of Object

- wlException (see wlException (object) on page 306)

Description

Sends a message to the Log Window that includes the error message and severity level stored in this `wlException` object.

Syntax

```
ReportLog()
```

Example

```
myUserException.ReportLog()
```

See also

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException() (see *wlException() (constructor)* on page 308)

RequestRetries (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Read operations are performed in a loop until all information is read. If the duration of this loop exceeds a specified timeout (see *ReceiveTimeout (property)* on page 215) or a socket error occurs, the Virtual Client will then retry to establish a connection. RequestRetries is the maximum number of times that the Virtual Client will attempt to reconnect to the server. The default value of RequestRetries is **9**.

Example

```
wlGlobals.RequestRetries = 7
```

See also

- *HTTP Components* (on page 24)
- ReceiveTimeout (see *ReceiveTimeout (property)* on page 215)

Reset() (method)

Method of Object

- wlOutputFile() (see *wlOutputFile (object)* on page 323)

Description

Return to the beginning of the output file.

Syntax

```
Reset()
```

Example

```
MyFileObj = new wlOutputFile(filename)
…
MyFileObj.Reset()
```

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)

- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- wlOutputFile() (see *wlOutputFile() (constructor)* on page 324)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

ResponseContentType (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The language in which WebLOAD receives the response from the SUT. This can be `HTML`, `XML`, or `Not Defined` and is used to decide whether or not to parse the response. If `ResponseContentType` is set to `HTML` or `XML`, the content of the response is treated as either HTML or XML, as specified. If not, an algorithm checks the content type of the response. This algorithm uses two other wlGlobals: HtmlContentTypes and XmlContentTypes. These contain a list of content types that specify HTMLs and XMLs, respectively. The algorithm checks whether the content type of the response matches either of these lists.

Syntax

```
wlGlobals.ResponseContentType = "TextString"
```

Example

```
wlGlobals.ResponseContentType = "HTML"
```

See also

- *HTTP Components* (on page 24)

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 script.

`RoundNum` does not exist in the global context of a script (`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 script.

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 scripts in a single Load Generator, WebLOAD maintains an independent round counter for each script. For example, if **WLFileAgendaShared** GUI mode

WebLOAD recommends accessing global system variables, including the `RoundNum` identification variable, through the WebLOAD Recorder. The variables that appear in this list are available for use at any point in a script file. In the WebLOAD Recorder main window, click **Variables Windows** in the **Debug** tab of the ribbon..

For example, it is convenient to add `RoundNum` to a Message Node to clarify the round in which the messages that appear in the WebLOAD Console Log window originated.

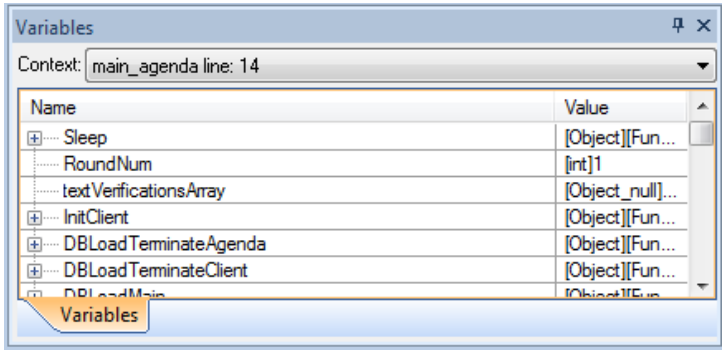

Figure 11: WebLOAD Recorder Variables Window

Note: `RoundNum` can also be added directly to the code in a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

See also

- ClientNum (see *ClientNum (variable)* on page 50)
- GeneratorName() (see *GeneratorName() (function)* on page 101)
- GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131)
- *Identification Variables and Functions* (on page 29)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- VCUniqueID() (see *VCUniqueID() (function)* on page 296)

row (object)

Property of Objects

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

- wlTables (see *wlTables (object)* on page 333)

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

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

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

Each `row` object contains information about the data found in the cells of a single table row. The `row` object includes the following properties:

- cell (see *cell (object)* on page 44) (`row` property)
- rowIndex (see *rowIndex (property)* on page 224) (`row` property)

Comment

```
The row object may be accessed as a member of the wlTables family of
table, row, and cell objects.
```

See also

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (wlTables property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (wlTables property)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell property)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- tagName (see *tagName (property)* on page 279) (cell property)

rowIndex (property)

Property of Object

- row (see *row (object)* on page 223)

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`.

Comment

The `rowIndex` property is a member of the `wlTables` family of table, row, and cell objects.

See also

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (wlTables property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (wlTables property)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell property)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (wlTables property)
- tagName (see *tagName (property)* on page 279) (cell property)
- wlTables (see *wlTables (object)* on page 333)

SaveHeaders (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Instruct WebLOAD to store the HTML response headers in `wlHeaders`.

- **false** –Do not store the header. (default)
- **true** – Store the header in `document.wlHeaders`.

Note: This property can only be inserted manually.

See also

- *HTTP Components* (on page 24)

SaveSource (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Instruct WebLOAD to store the complete HTML source code downloaded in an HTTP command.

- **false** – Do not store the source HTML (default).
- **true** – Store the source HTML in `document.wlSource`.

If you enable `SaveSource`, WebLOAD automatically stores the downloaded HTML whenever the script calls the `wlHttp.Get()` or `wlHttp.Post()` method. WebLOAD stores the most recent download in the `document.wlSource` property, refreshing it when the script calls `wlHttp.Get()` or `wlHttp.Post()` again. The stored code includes any scripts or other data embedded in the HTML. Your script can retrieve the code from `document.wlSource` and interpret it in any desired way.

See also

- *HTTP Components* (on page 24)

SaveTransaction (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Instruct WebLOAD to save detailed information about all transactions, both successes and failures, for later analysis in the Data Drilling reports.

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 disk space.

Note: Transaction data is only saved for the number of instances defined in the Instance limit field in the WebLOAD Console (Global Options dialog box, in the Data Drilling tab).

Possible values of the SaveTransaction property are:

- **false** – Do not store detailed data on successful transactions (default).
- **true** – Store detailed data on successful transactions.

The SaveTransaction property works with the following parameters in the Functional Testing tab (Automatic Data Collection area) of the Current Session Options/Default Options dialog box in the WebLOAD Console, as follows:

- **Pages** – When selected, WebLOAD provides timers and counters for every "Get" in the session. When SaveTransaction is set to true, WebLOAD provides an aggregate breakdown for all objects in the page.
- **Object level** – When selected, WebLOAD provides timers and counters for every object in every page. When SaveTransaction is set to true, WebLOAD provides breakdown information for every object in every page.
- **HTTP level** – When selected, WebLOAD provides breakdown information for every failed transaction in every page. When SaveTransaction is set to true, WebLOAD provides breakdown information for every instance of every failed transaction in every page.

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

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- *HTTP Components* (on page 24)

script (object)

Property of Object

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 (see *document (object)* on page 78)

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 (see *event (property)* on page 92)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- language (see *language (property)* on page 161)
- src (see *src (property)* on page 252)

See also

- *Collections* (on page 27)

search (property)

Property of Objects

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

Description

The search attribute string of the URL, not including the initial ? symbol (read-only string).

Example

```
"SearchFor=modems&SearchType=ExactTerm"
```

Seed() (method)

Method of Object

- wlRand (see *wlRand (object)* on page 326)

Description

Initialize the random number generator. Call the `Seed()` method in the `InitAgenda()` function of a script, using any integer as a `seed`.

Syntax

```
wlRand.Seed(seed)
```

Parameters

Parameter Name	Description
`seed`	Seed integer.

Example

```
wlRand.Seed(12345)
```

See also

- Num() (see *Num() (method)* on page 177)
- Range() (see *Range() (method)* on page 215)
- Select() (see *Select() (method)* on page 230)

Select

Select() (method)

Method of Object

- wlRand (see *wlRand (object)* on page 326)

Description

Select one element of a set at random.

Syntax

```
wlRand.Select(val1, val2, ..., valN)
```

Parameters

Parameter Name	Description
val1…val*N*	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() (see *Num() (method)* on page 177)
- Range() (see *Range() (method)* on page 215)
- Seed() (see *Seed() (method)* on page 229)

SelectSecondTimeout (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Specify a second option for the amount of time the system will wait for a TCP connection to be established before timing out. The default value of SelectSecondTimeout is **0 milliseconds**.

Syntax

```
wlGlobals.SelectSecondTimeout = number
```

Example

```
wlGlobals.SelectSecondTimeout = 100
```

See also

- SelectSwitchNum (see *SelectSwitchNum (property)* on page 231)
- SelectTimeout (see *SelectTimeout (property)* on page 232)

SelectSwitchNum (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The number of iterations after which the timeout is made shorter and the sleep time longer to ease the stress on the CPU. The default value of SelectSwitchNum is **100**.

Syntax

```
wlGlobals.SelectSwitchNum = number
```

Example

```
wlGlobals.SelectSwitchNum = 500
```

Comment

The following describes the basic functionality of the Load Engine.

Each read operation is limited by the RecvTimeout property. That is, the entire algorithm described below exits when it reaches the RecvTimeout timeout.

First, a send operation is performed with a request to the server:

```
numberOfIterations = 0
while (wlGlobals.SendTimeout not exceeded) {
  if (numberOfIterations < wlGlobals.SelectWriteSwitchNum {
     timeout = wlGlobals.SelectWriteTimeout
     sleepTime = wlGlbobals.SelectWriteSecondTimeout
  }
  else {
     timeout = wlGlobals.SelectWriteSecondTimeout
```

```
      sleepTime = wlGlbobals.SelectWriteTimeout
   }
   select for read (timeout)
   if (socket ready) break
}
send request
```

Then the Load Engine attempts to read the response:

```
numberOfIterations = 0
while (wlGlobals.RecvTimeout not exceeded) {
   if (numberOfIterations < wlGlobals.SelecSwitchNum {
      timeout = wlGlobals.SelecTimeout
      sleepTime = wlGlbobals.SelectSecondTimeout
   }
   else {
      timeout = wlGlobals.SelectSecondTimeout
      sleepTime = wlGlbobals.SelectTimeout
   }
   select for write (timeout)
   if (socket ready) break
}
read response
```

The IO operations of the Load Engine are synchronous, basically looping and polling the sockets. The Load Engine performs a select on the socket with a timeout. If the select on the socket fails, a short sleep is performed and the system tries to perform the select again until the timeout specified in the SelectTimeout property is exceeded. This also enables the Load Engine to check with the monitor between selects to see if the session has ended.

See also

- SelectSecondTimeout (see *SelectSecondTimeout (property)* on page 230)
- SelectTimeout (see *SelectTimeout (property)* on page 232)

SelectTimeout (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

A timeout used when performing a select on a socket. If the select on the socket fails, a short sleep is performed and the system tries to perform the select again until the timeout specified in the SelectTimeout property is exceeded. The default value of SelectTimeout is **200 milliseconds**.

Syntax

```
wlGlobals.SelectTimeout = number
```

Example

```
wlGlobals.SelectTimeout = 300
```

See also

- SelectSecondTimeout (see *SelectSecondTimeout (property)* on page 230)
- SelectSwitchNum (see *SelectSwitchNum (property)* on page 231)

SelectWriteSecondTimeout (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Specify a second option of the amount of time that the connection should stay open while nothing is being written to it. The default value of SelectWriteSecondTimeout is **10 milliseconds**.

Syntax

```
wlGlobals.SelectWriteSecondTimeout = number
```

Example

```
wlGlobals.SelectWriteSecondTimeout = 100
```

See also

- SelectWriteSwitchNum (see *SelectWriteSwitchNum (property)* on page 234)
- SelectWriteTimeout (see *SelectWriteTimeout (property)* on page 234)

SelectWriteSwitchNum (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Indicate the amount of time that elapses before switching from the first write timeout to the second write timeout. The default value of SelectWriteSwitchNum is **100 milliseconds**.

Syntax

```
wlGlobals.SelectWriteSwitchNum = number
```

Example

```
wlGlobals.SelectWriteSwitchNum = 100
```

See also

- SelectWriteSecondTimeout (see *SelectWriteSecondTimeout* on page 233)
- SelectWriteTimeout (see *SelectWriteTimeout* on page 234)

SelectWriteTimeout (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Specify the amount of time that the connection should stay open while nothing is being written to it. The default value of SelectWriteTimeout is **200 milliseconds**.

Syntax

```
wlGlobals.SelectWriteTimeout = number
```

Example

```
wlGlobals.SelectWriteTimeout = 300
```

See also

- SelectWriteSecondTimeout (see *SelectWriteSecondTimeout (property)* on page 233)

- SelectWriteSwitchNum (see *SelectWriteSwitchNum (property)* on page 234)

selected (property)

Property of Objects

- option (see *option (object)* on page 185)

Description

The `selected` property has a value of `true` if this <OPTION> element has been selected, or `false` otherwise (read-only).

See also

- location (see *location (object)* on page 168)

selectedindex (property)

Property of Objects

- element (see *element (object)* on page 80)
- location (see *location (object)* on page 168)

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**.

SendBufferSize (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The `SendBufferSize` property defines the amount of space allocated to the outgoing data buffer. The default value of SendBufferSize is **-1**, which indicates that the entire request should be sent in one buffer.

Note: This property can only be inserted manually.

Syntax

```
wlGlobals.SendBufferSize = number
```

Example

```
wlGlobals.SendBufferSize = 300
```

SendClientStatistics (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

SendClientStatistics is used to define whether or not to send statistics. It should be set in InitAgenda. The console writes the raw data to `C:\Documents and Settings\<user name>\Local Settings\Temp\ClientStat.txt`. To change the name, add `CLIENT_STATISTICS_FILE = "file-name"` to `webload.ini` on the console side. By default, this will save the raw statistics data for all the clients.

Only statistics calculated by the Load Generator are supported. Statistics like hits/second are not sent and empty statistics, for example, RoundTime where no round was completed within a time slice, are sent as -1.

Example

```
wlGlobals.SendClientStatistics = true
```

See also

- SendClientStatisticsFilter (see *SendClientStatisticsFilter (property) on* page 236

SendClientStatisticsFilter (property)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

This is a very simple filter for choosing clients for which to send statistics. The format is "1;4;8;". Each number is a client number and you can specify as many client numbers as you want. Ranges and wildcards are not supported. If no client numbers are specified, statistics are sent for all clients.

If spawning is being used and you specify, for example, "1", statistics for the first VC will be sent for each slave process.

Only statistics calculated by the Load Generator are supported. Statistics like hits/second are not sent and empty statistics, for example, RoundTime where no round was completed within a time slice, are sent as -1.

Note: The filter and file name are not checked.

Syntax

```
wlGlobals.SendClientStatisticsFilter = "<client number>;<client number>;"
```

Example

```
wlGlobals.SendClientStatisticsFilter = "2;3;8"
```

See also

- SendClientStatistics (see *SendClientStatistics (property)* on page 236)

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 a script.

Syntax

```
SendCounter(EventName)
```

Parameters

Parameter Name	Description
`EventName`	A string with the name of the event being counted.

See also

- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)
- Sleep() (see *Sleep() (function)* on page 248)

- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)

SendMeasurement() (function)

Description

Use this function to assign a value to the specified statistical measurement. Call `SendMeasurement()` in the main script of a script.

Syntax

```
SendMeasurement(MeasurementName, value)
```

Parameters

Parameter Name	Description
`MeasurementName`	A string with the name of the measurement being set.
`value`	An integer value to set.

Example

```
NumberOfImagesInPage = document.images.length
SendMeasurement("NumberOfImagesInPage",NumberOfImagesInPage)
```

GUI mode

WebLOAD recommends setting measurement functions within script files directly through the WebLOAD Recorder. In WebLOAD Recorder, drag the **Send Measurement** icon from the Load toolbox into the Script Tree at the desired location. The Send Measurement dialog box opens. Select a measurement name and its value and click **OK**. The **Send Measurement** item appears in the Script Tree and the JavaScript code is added to the script. To see the new JavaScript code, view the script in JavaScript Editing mode.

See also

- SendCounter() (see *SendCounter() (function)* on page 237)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)
- Sleep() (see *Sleep() (function)* on page 248)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)

SendTimer() (function)

Description

Use this function to output the value of a timer to the WebLOAD Console. Call `SendTimer()` in the main script of a script, 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

Parameter Name	Description
`TimerName`	A string with the name of the timer being sent to the WebLOAD Console.

Example

```
SendTimer("Link 3 Time")
```

GUI mode

WebLOAD recommends setting timer functions within script files directly through the WebLOAD Recorder. In WebLOAD Recorder, drag the Send Timer  icon from the Load toolbox into the Script Tree at the desired location. The Send Timer dialog box opens. Enter a timer name and click **OK**. The Send Timer item appears in the Script Tree and the JavaScript code is added to the script. To see the new JavaScript code, view the script in JavaScript Editing mode.

See also

- SendCounter() (see *SendCounter() (function)* on page 237)
- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SetTimer() (see *SetTimer() (function)* on page 245)
- Sleep() (see *Sleep() (function)* on page 248)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)

Set() (method)

Set() (addition method)

Method of Objects

- wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309)
- wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)

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

Parameter Name	Description
`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 script, 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 scripts, 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 script, 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 scripts, run by all processes, on all Load Generators, system-wide.

Example

```
wlGeneratorGlobal.Set("MySharedCounter", 0, WLCurrentAgenda)
wlSystemGlobal.Set("MyGlobalCounter", 0, WLCurrentAgenda)
```

See also

- Add() (see *Add() (method)* on page 39)
- Get() (see *Get() (addition method)* on page 102)

Set() (cookie method)

Method of Object

- location (see *location (object)* on page 168)
- wlCookie (see *wlCookie (object)* on page 302)

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

Parameter Name	Description
`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`".

Comment

Set cookies within the main script of the script. 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`.

SetClientType (function)

Description

The HTTP client has the following sub types, which can be set using the SetClientType function. These sub-types are:

- **Normal** (default) – When the client type is set to Normal, a DOM is created without tables.
- **Thick** – When the client type is set to Thick, the tables structure is included in the DOM.
- **Thin** – When the client type is set to Thin, no DOM or headers are created, and each page is parsed only once. This type is used for very high performance with static pages.

Use the SetClientType function with the Thick sub-type when you want to parse tables or with the Thin sub-type when you want optimize tests for simple writes with static pages.

Syntax

```
setClientType(clientType)
```

Parameters

Parameter Name	Description
`clientType`	A supplied string that identifies the client type. The possible client types are Normal, Thin, Thick, Custom.

Example

```
setClientType(thick)
```

Comment

When you call SetClientType("Thin"), the ParseOnce flag is set to true. Each page will only be parsed the first time it is read and the list of all the resources it accesses will be

saved. The next time the page is needed, the list will be reused and no additional parsing will be performed.

See also

- *HTTP Components* (on page 24)
- ParseOnce (see *ParseOnce (property) on* page 198)
- GetImagesInThinClient (see *GetImagesInThinClient (property)* on page 122)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

SetFailureReason() (function)

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

Parameter Name	Description
`ReasonName`	A user-supplied string that identifies and categorizes the reason for this transaction instance failure.

Comment

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

- CreateDOM() (see *CreateDOM() (function)* on page 63)
- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

setTimeout() (function)

Description

Execute the specified callback after a specified number of milliseconds.

The script execution will continue immediately, not waiting for the specified time or the function to execute (unlike the *Sleep() (function)*, which instructs the script to pause for a specified time).

Syntax

```
setTimeout (Func, PauseTime)
```

Parameters

Parameter Name	Description
Func	Function to be called after the specified number of milliseconds
PauseTime	An integer value specifying the number of milliseconds to pause.

Example

To pause for 1 second, write:

```
InfoMessage("before setTimeout");
setTimeout( function() {
   InfoMessage("in setTimeout");
} , 1000);
InfoMessage("after setTimeout");
Sleep(2000);
InfoMessage("after sleep");
```

See also

- *Sleep() (function)* (on page 248)

SetTimer() (function)

Description

Use this function to zero a timer. Call `SetTimer()` in the main script of a script, 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 script; the timer continues running between rounds if you do not zero it.

Syntax

```
SetTimer(TimerName)
```

Parameters

Parameter Name	Description
`TimerName`	A string with the name of the timer being zeroed.

Example

```
SetTimer("Link 3 Time")
```

GUI mode

WebLOAD recommends setting timer functions within script files directly through the WebLOAD Recorder. In WebLOAD Recorder, drag the Set Timer icon from the Load toolbox into the Script Tree at the desired location. The Set Timer dialog box opens. Enter a timer name and click **OK**. The Set Timer item appears in the Script Tree and the JavaScript code is added to the script. To see the new JavaScript code, view the script in JavaScript Editing mode.

See also

- SendCounter() (see *SendCounter() (function)* on page 237)
- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SendTimer() (see *SendTimer() (function)* on page 239)
- Sleep() (see *Sleep() (function)* on page 248)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)

SevereErrorMessage() (function)

Description

Use this function to display a severe error message in the Log Window of the WebLOAD Console, stop the session, and abort the Load Generator.

Syntax

```
SevereErrorMessage(msg)
```

Parameters

Parameter Name	Description
`msg`	A string with a severe error message to be sent to the WebLOAD Console.

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 script. You may also use the wlException (see *wlException (object)* on page 306) object with the built-in `try()/catch()` commands to catch errors within your script. For more information about error management and execution sequence options, see *Error Management* in the *WebLOAD Scripting Guide*.

GUI mode

WebLOAD recommends adding message functions to your script files directly through the WebLOAD Recorder. Drag the Message icon from the WebLOAD Recorder toolbox into the script. The Message dialog box opens. Enter the message text, select a severity level for the message, and click **OK**.

See also

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)
- wlException (see *wlException (object)* on page 306)
- wlException() (see *wlException() (constructor)* on page 308)

Severity (property)

Property of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

`Severity` is used to define the global severity of a verification fail error. When defined, `Severity` affects all the verifications in which severity is not defined. If you define the error severity for a specific verification, it overrides the global severity defined in the `Severity` property.

Possible values of the `Severity` property are:

- `WLSuccess` – The transaction terminated successfully.
- `WLMinorError` – This specific transaction failed, but the test session may continue as usual. The script 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 script 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 script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

To set the global severity of all verification fail errors to `WLError`, write:

```
wlVerification.Severity = WLError
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- PageTime (see *PageTime (property)* on page 190)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

Size (property)

Property of Object

- element (see *element (object)* on page 80)

- *Select* (on page 230)

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.

Sleep() (function)

Description

Pause for a specified number of milliseconds.

Syntax

```
Sleep(PauseTime)
```

Parameters

Parameter Name	Description
`PauseTime`	An integer value specifying the number of milliseconds to pause.

Example

To pause for 1 second, write:

```
Sleep(1000)
```

GUI mode

WebLOAD recommends setting sleep functions within script files directly through the WebLOAD Recorder. Drag the **Sleep** icon from the General toolbox into the Script Tree at the desired location. The Sleep dialog box opens. Enter or select the duration of the sleep and click **OK**. The Sleep item appears in the Script Tree and the JavaScript code is added to the script.

Sleep function command lines may also be added directly to the code in a JavaScript Object within a script through the IntelliSense Editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

Comment

Specify one of the sleep options when running a test script in the Sleep Time Control tab, through the WebLOAD Recorder from the **Tools ➤ Default** or **Current Project Options** or through the WebLOAD Console from the **Tools ➤ Default** or **Current Project Options**:

- **Sleep time as recorded** – Runs the script with the delays corresponding to the natural pauses that occurred when recording the script.

- **Ignore recorded sleep time (default)** – Eliminates any pauses when running the script and runs a worst-case stress test.
- **Set random sleep time** – Sets the ranges of delays to represent a range of users.
- **Set sleep time deviation** – Sets the percentage of deviation from the recorded value to represent a range of users.

For more information on setting the Sleep Time Control settings, see *Configuring Sleep Time Control Options* in the *WebLOAD Recorder User's Guide*.

See also

- DisableSleep (see *DisableSleep (property)* on page 76)
- SendCounter() (see *SendCounter() (function)* on page 237)
- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)
- SleepDeviation (see *SleepDeviation (property)* on page 249)
- SleepRandomMax (see *SleepRandomMax (property)* on page 250)
- SleepRandomMin (see *SleepRandomMin (property)* on page 251)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)
- *Using the IntelliSense JavaScript Editor* (on page 18)

SleepDeviation (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Integer that indicates the percentage by which recreated sleep periods should deviate from the original recorded time.

Example

```
wlGlobals.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

In WebLOAD Recorder, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Console, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Session Options** dialog box or the **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- DisableSleep (see *DisableSleep (property)* on page 76)
- Sleep() (see *Sleep() (function)* on page 248)
- SleepRandomMax (see *SleepRandomMax (property)* on page 250)
- SleepRandomMin (see *SleepRandomMin (property)* on page 251)

SleepRandomMax (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Integer that indicates the maximum length of a recreated sleep period when not using the original recorded time.

Syntax

```
wlGlobals.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

In WebLOAD Recorder, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Console, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Session Options** dialog box or in the **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- DisableSleep (see *DisableSleep (property)* on page 76)
- Sleep() (see *Sleep() (function)* on page 248)
- SleepDeviation (see *SleepDeviation (property)* on page 249)
- SleepRandomMin (see *SleepRandomMin (property)* on page 251)

SleepRandomMin (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

Description

Integer that indicates the minimum length of a recreated sleep period when not using the original recorded time.

Syntax

```
wlGlobals.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

In WebLOAD Recorder, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

In WebLOAD Console, select the sleep mode in the Sleep Time Control tab of the **Default** or **Current Session Options** dialog box or in the **Script Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- DisableSleep (see *DisableSleep (property)* on page 76)
- Sleep() (see *Sleep() (function)* on page 248)
- SleepDeviation (see *SleepDeviation (property)* on page 249)
- SleepRandomMax (see *SleepRandomMax (property)* on page 250)

src (property)

Property of Object

- Image (see *Image (object)* on page 149)
- script (see *script (object)* on page 228)

- wlXmls (see *wlXmls (object)* on page 340)

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"
```

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- load() (see *load() (method)* on page 163)
- loadXML() (see *loadXML() (method)* on page 167)
- *load() and loadXML() Method Comparison* (on page 164)
- XMLDocument (see *XMLDocument (property)* on page 345)

SSLBitLimit (property)

Property of Object

- wlGlobals (see *wlGlobals (object)* on page 313)

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 (see *SSLCryptoStrength (property)* on page 258) to define and enable cipher levels and cryptographic strengths using a higher, more categorical approach.

Note: This property can only be inserted manually.

Syntax

```
wlGlobals.SSLBitLimit = IntegerValue
```

Example

```
wlGlobals.SSLBitLimit = 56
```

-Or-

```
CurrentBitLimit = wlGlobals.SSLBitLimit
```

GUI mode

WebLOAD recommends setting the SSL bit limit through the WebLOAD Console. Check SSL Bit Limit and select a value from the drop-down list on the SSL tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLEnableStrength() (see *SSLEnableStrength() (function)* on page 262)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlHttp (see *wlHttp (object)* on page 316)

SSLCipherSuiteCommand() (function)

Description

Set the SSL configuration environment before running a test script.

Note: This property can only be inserted manually.

Syntax

```
SSLCipherSuiteCommand("SSLCipherCommand")
```

Parameters

Parameter Name	Description
`SSLCipherCommand`	One of the following commands, used to set the SSL configuration environment before running a test script. • `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) Note that the command name should appear in quotes.

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 `InitClient()` function.

```
InitClient()
{
  …
  SSLCipherSuiteCommand("DisableAll")
  SSLEnableCipherName("EXP-RC4-MD5")
  …
}
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)

- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

SSLClientCertificateFile, SSLClientCertificatePassword (properties)

Properties of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

SSL Client certificates offer a more secure method of authenticating users in an Internet commerce scenario then 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 script 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.

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 WebLOAD Console. Enter user authentication information through the Authentication tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) `(wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) `(wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)

- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)

SSLCryptoStrength (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)

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 script'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 ciphers:

- `DHE_DSS_ RC4_SHA`
- `DES_CBC_MD5`
- `AECDH_NULL_SHA`
- `EXP_RC4_MD5`

If you then set `SSLCryptoStrength` to `SSL_ExportCryptoOnly`, then only the last two ciphers, `AECDH_NULL_SHA`, and `EXP_RC4_MD5`, will be enabled.

```
InitClient()
{
  ?
  SSLEnableCipherName("DHE_DSS_RC4_SHA")
  SSLEnableCipherName("DES_CBC_MD5")
  SSLEnableCipherName("AECDH_NULL_SHA")
  SSLEnableCipherName("EXP_RC4_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

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)

SSLDisableCipherID() (function)

Description

Disables the specified SSL cipher for the current session.

Syntax

```
SSLDisableCipherID(CipherID)
```

Parameters

Parameter Name	Description
CipherID	The SSL cipher to disable for the current session.

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 `InitClient()` function.

```
Initclient()
{
  …
  SSLDisableCipherID(45)
  …
}
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)

- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

SSLDisableCipherName() (function)

Description

Disables the specified SSL cipher for the current session.

Syntax

```
SSLDisableCipherName(CipherName)
```

Parameters

Parameter Name	Description
`CipherName`	Any of the SSL protocol names. See *WebLOAD-supported SSL Protocol Versions* (on page 449) for a complete list of protocol names.

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 `InitClient()` function.

```
InitClient()
{
  …
  SSLDisableCipherName("EXP-RC4-MD5")
  …
}
```

See also

- *Browser Configuration Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)

- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (wlGlobals only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

SSLEnableStrength() (function)

Description

Enables all ciphers which have encryption strength not greater than specified by the parameter. Function allows you to limit SSL key length without iterating over the whole ciphers list.

Syntax

```
SSLEnableStrength(MaxStrength)
```

Parameters

Parameter Name	Description
`MaxStrength`	The maximum encryption strength allowed. Strength must be specified in bits (typical values are 40, 56, 96, 128, 168, 196, 256, 512, 1024, etc.).

Example

Your test session may include a variety of function calls related to specific set of SSL ciphers. For example, you may wish to test your application with weak ciphers only.

The following `InitAgenda()` function fragment enables all protocols with encryption strength less or equal to 128 bits.

```
InitAgenda()
{
   …
   SSLEnableStrength(128)
   …
}
```

See also

- *Browser Configuration Components* (on page 24)
- SSLBitLimit (see "*SSLBitLimit (property)*" on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see "*SSLCipherSuiteCommand() (function)*" on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see "*SSLClientCertificateFile, SSLClientCertificatePassword (properties)*" on page 256)
- SSLCryptoStrength (see "*SSLCryptoStrength (property)*" on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see "*SSLDisableCipherID() (function)*" on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherName() (see "*SSLEnableCipherName() (function)*" on page 265)
- SSLGetCipherCount() (see "*SSLGetCipherCount() (function)*" on page 266)
- SSLGetCipherID() (see "*SSLGetCipherID() (function)*" on page 267)
- SSLGetCipherInfo() (see "*SSLGetCipherInfo() (function)*" on page 269)
- SSLGetCipherName() (see "*SSLGetCipherName() (function)*" on page 270)
- SSLGetCipherStrength() (see "*SSLGetCipherStrength() (function)*" on page 271)
- SSLUseCache (see "*SSLUseCache (property)*" on page 272)
- SSLVersion (see "*SSLVersion (property)*" on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

SSLEnableCipherID() (function)

Description

Enables the specified SSL cipher for the current session.

Syntax

```
SSLEnableCipherID(CipherID)
```

Parameters

Parameter Name	Description
`CipherID`	Any of the SSL protocol ID numbers. Use `SSLGetCipherID()` (see *SSLGetCipherID() (function)* on page 267) function to get the ID number associated with a specified protocol name. See *WebLOAD-supported SSL Protocol Versions* (on page 449) for a complete list of protocol names.

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 script 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 script. The following `InitClient()` function fragment disables all protocols, gets a protocol ID number, and enables the selected protocol in the `InitClient()` function.

```
InitClient()
{
  …
  SSLCipherSuiteCommand(DisableAll)
  MyCipherID = SSLGetCipherID("EXP-RC4-MD5")
  SSLEnableCipherID(MyCipherID)
  …
}
```

See also

- *Browser Configuration Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)

- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

SSLEnableCipherName() (function)

Description

Enables the specified SSL cipher for the current session.

Syntax

```
SSLEnableCipherName(CipherName)
```

Parameters

Parameter Name	Description
`CipherName`	Any of the SSL protocol names. See *WebLOAD-supported SSL Protocol Versions* (on page 449) for a complete list of protocol names.

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("EXP-RC4-MD5")
  …
}
```

See also

- *Browser Configuration Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)

SSLGetCipherCount() (function)

Description

Returns an integer, the number of ciphers enabled for the current test session. While that may seem obvious if your script 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()
```

Return Value

Returns an integer representing the number of ciphers enabled for the current test session.

Example

```
CurrentCipherCount = SSLGetCipherCount()
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- wlHttp (see *wlHttp (object)* on page 316)

SSLGetCipherID() (function)

Description

Returns the ID number associated with the specified cipher.

Syntax

```
SSLGetCipherID(CipherName)
```

Parameters

Parameter Name	Description
`CipherName`	Any of the SSL protocol names. See *WebLOAD-supported SSL Protocol Versions* (on page 449) for a complete list of protocol names.

Return Value

Returns the ID number associated with the specified cipher.

Example

```
MyCipherID = SSLGetCipherID("EXP-RC4-MD5")
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

SSLGetCipherInfo() (function)

Description

Prints a message on the WebLOAD Console with information about the specified SSL protocol.

Syntax

```
SSLGetCipherInfo(CipherName ➤ CipherID)
```

Parameters

Accepts either one of the following parameters:

Parameter Name	Description
`CipherName`	The name of the SSL cipher.
`CipherID`	The identification number of the SSL cipher.

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("EXP-RC4-MD5")
```

-Or-

```
SSLGetCipherInfo(2)
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)

- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

SSLGetCipherName() (function)

Description

Returns the name of the cipher associated with the specified ID number.

Syntax

```
SSLGetCipherName(CipherID)
```

Parameters

Parameter Name	Description
`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

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)

- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

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()
```

Return Value

Returns an integer representing the maximum available cipher strength for the current session.

Example

```
CurrentCipherStrength = SSLGetCipherStrength()
```

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)

- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLUseCache (see *SSLUseCache (property)* on page 272)
- SSLVersion (see *SSLVersion (property)* on page 274)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

SSLUseCache (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Enable caching of SSL decoding keys received from an SSL (HTTPS) server. The value of `SSLUseCache` may be:

- **false** – Disable caching.
- **true** – Enable caching. (default)

A `true` 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 `true` value to reduce transmission time during SSL communication. Assign a `false` 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.

GUI mode

WebLOAD recommends enabling or disabling the SSL cache through the WebLOAD Console. Enable caching for the Load Generator or for the Probing Client during a test session by checking the appropriate box in the Browser Parameters tab of the **Default Options** dialog box, accessed from the **Tools** tab of the ribbon.

Comment

To clear the SSL cache, set the ClearSSLCache() (see *ClearSSLCache() (method)* on page 49) property.

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLVersion (see *SSLVersion (property)* on page 274)

SSLVersion (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

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 *WebLOAD-supported SSL Protocol Versions* (on page 449) for a table illustrating all the Client/Server SSL version handshake combination possibilities and a complete list of SSL/TLS protocol names.

See also

- *HTTP Components* (on page 24)
- SSLBitLimit (see *SSLBitLimit (property)* on page 253) (`wlGlobals` only)
- *SSL Cipher Command Suite* (on page 33)
- *SSL Ciphers – Complete List* (on page 450)
- SSLCipherSuiteCommand() (see *SSLCipherSuiteCommand() (function)* on page 255)
- SSLClientCertificateFile, SSLClientCertificatePassword (see *SSLClientCertificateFile, SSLClientCertificatePassword (properties)* on page 256)
- SSLCryptoStrength (see *SSLCryptoStrength (property)* on page 258) (`wlGlobals` only)
- SSLDisableCipherID() (see *SSLDisableCipherID() (function)* on page 260)
- SSLDisableCipherName() (see *SSLDisableCipherName() (function)* on page 261)
- SSLEnableCipherID() (see *SSLEnableCipherID() (function)* on page 264)
- SSLEnableCipherName() (see *SSLEnableCipherName() (function)* on page 265)
- SSLGetCipherCount() (see *SSLGetCipherCount() (function)* on page 266)
- SSLGetCipherID() (see *SSLGetCipherID() (function)* on page 267)
- SSLGetCipherInfo() (see *SSLGetCipherInfo() (function)* on page 269)
- SSLGetCipherName() (see *SSLGetCipherName() (function)* on page 270)
- SSLGetCipherStrength() (see *SSLGetCipherStrength() (function)* on page 271)
- SSLUseCache (see *SSLUseCache (property)* on page 272)

StopClient () (function)

Description

Stops the execution of the Virtual Client running the script from which `StopClient()` was called. After `StopClient()` is called, this client cannot be resumed.

Syntax

```
StopClient ([SeverityLevel], [Reason])
```

Parameters

Parameter Name	Description
`SeverityLevel`	Optionally, specify the severity level of the error that occurred. The possible values are: • `WLMinorError`. The message is displayed as a warning message. • `WLError`. The message is displayed as an error message. If no severity level is specified, `WLMinorError` is assumed. **Note:** Error levels are used for display in the log window and do not define any logical behavior.
`Reason`	An optional string containing the reason for stopping the virtual client running the script. If no reason is specified, a default message is displayed. See *Default Message* below.

Default Message

The following default message is displayed when no reason is specified:

```
StopClient(WLError, "Client"+%d+"was terminated by a user command in the script")
```

where `%d` is a parameter that takes the number of the client that was terminated. For example, "Client 38 was terminated by a user command in the script".

Examples

```
StopClient(WLError, "Error occurred when running script, client terminated")
StopClient(, "Error occurred when running script, client terminated")
StopClient(WLError)
```

StopGenerator () (function)

Description

Stop the script from within the script. The string passed as a parameter is the message that appears when the script is stopped.

Syntax

```
StopGenerator(string)
```

Parameters

Parameter Name	Description
String	The message to be displayed when the script is stopped.

Example

```
StopGenerator("The script was terminated from within the script")
```

string (property)

Property of Object

- title (see *title (property)* on page 284)

Description

Stores the document title in a text string.

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 a script. 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 script simultaneously. For more information on Synchronization Points, see *Working with Synchronization Points* in the *WebLOAD Scripting Guide*.

Syntax

```
SynchronizationPoint([timeout])
```

Parameters

Parameter Name	Description
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 script 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

The following script 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("Syncronization failed, Load Size changed")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
if (SP == WLTimeout)
{
  InfoMessage("Syncronization failed, Timeout expired")
  InfoMessage("SP = " + SP.toString() + " " + ClientNum)
}
if (SP == WLError)
{
  InfoMessage("Syncronization 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 script files directly through the WebLOAD Recorder. Drag the **Synchronization Point** icon from the Load toolbox into the Script Tree directly before the action you want all Virtual Clients to perform simultaneously. The **Synchronization Point** dialog box opens. Enter or select a timeout value for the Synchronization Point and click **OK**. The **Synchronization Point** item appears in the Script Tree and the JavaScript code is added to the script. The JavaScript code line that corresponds to the Synchronization Point in the Script Tree appears in the JavaScript View.

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

- SendCounter() (see *SendCounter() (function)* on page 237)
- SendMeasurement() (see *SendMeasurement() (function)* on page 238)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)
- Sleep() (see *Sleep() (function)* on page 248)
- SynchronizationPoint() (see *SynchronizationPoint() (function)* on page 277)
- *Timing Functions* (on page 34)

tagName (property)

Property of Object

- cell (see *cell (object)* on page 44)

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
```

Comment

The `tagName` property is a member of the `wlTables` family of table, row, and cell objects.

See also

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- cols (see *cols (property)* on page 54) (wlTables property)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- id (see *id (property)* on page 146) (wlTables property)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell property)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- row (see *row (object)* on page 223) (wlTables property)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- wlTables (see *wlTables (object)* on page 333)

target (property)

Property of Object

- form (see *form (object)* on page 95)
- link (see *link (object)* on page 162)

Description

The name of the window or frame into which the form or link should be downloaded (read-only string).

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

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 the *WebLOAD Scripting 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

- wlTarget (see *wlTarget (property)* on page 334)

Text (function)

Description

Verify the absence or presence of a specified text expression within the Web server response.

Syntax

```
wlVerification.Text(searchOption, text, severity)
```

Parameters

Parameter Name	Description
`searchOption`	Possible values are: `WLFind` or `WLNotFind`.
`text`	String text to find in the document.wlSource.

`severity`	Possible values of this parameter are: • `WLSuccess`. The transaction terminated successfully. • `WLMinorError`. This specific transaction failed, but the test session may continue as usual. The script 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 script 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 script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

The following code verifies that the server response does not contain the word "error". In case of failure, WebLOAD displays the error message and stops the execution.

```
wlVerification.Text(WLNotFind, "error", WLError);
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

ThreadNum() (property)

Description

Assigns a unique number to a process based on the client/Load Generator/script. This number is unique across the script's slave processes and is saved in the ClientNum property. Each client in a Load Generator is assigned a unique number. However, two clients in two different Load Generators may have the same number.

Note: While `ClientNum` is unique within a single Load Generator, it is not unique system wide. Use VCUniqueID() (see *VCUniqueID() (function)* on page 296) 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, ..., N-1`. You can access `ClientNum` anywhere in the local context of the script

(`InitClient()`, main script, `TerminateClient()`, etc.). `ClientNum` does not exist in the global context (`InitAgenda()`, `TerminateAgenda()`, etc.).

If you mix scripts within a single Load Generator, instances of two or more scripts 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 (property)* on page 165).

Comment

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

Example

```
InfoMessage("ThreadNum: " + ThreadNum())
```

See also

- ClientNum() (see *ClientNum (property)* on page 50)
- VCUniqueID (see *VCUniqueID() (function)* on page 296)

TimeoutSeverity (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

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 WebLOAD Console. Check Verification in the Page Time area and select an error severity level from the drop-down box in the Functional Testing tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Syntax

You may also assign a severity level using the `TimeoutSeverity` property.

```
wlGlobals.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 script 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 script 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 script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.

Example

```
wlGlobals.TimeoutSeverity = WLError.
```

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- *Transaction Verification Components* (on page 36)
- TransactionTime (see *TransactionTime (property)* on page 287)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

title (property)

Property of Objects

- document (see *document (object)* on page 78)
- element (see *element (object)* on page 80)
- frames (see *frames (object)* on page 99)
- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

- script (see *script (object)* on page 228)

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 (see *string (property)* on page 277)

See also

- *Collections* (on page 27)
- form (see *form (object)* on page 95)
- *Select* (on page 230)

Title (function)

Method of Object

- wlVerification (see *wlVerification (object)* on page 337)

Description

This function enables you to validate a HTML Web page's title.

Syntax

```
wlVerification.Title(<ExpectedTitle>, <Severity>\<FunctionName>
[, <ErrorMessage>\<FunctionArguments>])
```

Parameters

Parameter Name	Description
ExpectedTitle	A user-supplied string that identifies the expected title of the HTML Web page. If the string you enter in this parameter appears in the HTML Web page's title, the validation is successful.
Severity	Possible values of this parameter are: • WLSuccess. The transaction terminated successfully. • WLMinorError. This specific transaction failed, but the test session may continue as usual. The script 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 script 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 script displays an error message in the Log window and the Load Generator on which the error occurred is stopped.
FunctionName	A pre-defined Javascript function that is called if the verification fails.
[ErrorMessage]	string
[FunctionArguments]	The arguments for the function that is called if the verification fails.

Example

For example: when validation fails, an email is sent with a message (errorMessage).

The function `sendEmailOnError` has the arguments `emailAddress` and `errorMessage`.

```
function sendEmailOnError(emailAddress, errorMessage)
{
  sendEmailTo(emailAddress, errorMessage)
}
```

When Title validation fails. The following function is called:

```
sendEmailOnError(VP@rrrr.com, "Title validation failed");
```

So the Title function syntax is as follows:

```
wlVerification.Title("compare title", sendEmailOnError, VP@rrrr.com,
"Title validation failed");
```

See also

- wlVerification (see *wlVerification (object)* on page 337)
- PageContentLength (see *PageContentLength (property)* on page 189)
- PageTime (see *PageTime (property)* on page 190)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)

TransactionTime (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

Description

Assign a timeout value using the TransactionTime property. Use the TransactionTime property to set a timeout limit for verification on the maximum transaction time.

GUI mode

WebLOAD recommends setting page verification timeout values through the WebLOAD Console. Check Page Verification and enter a maximum number of seconds in the Functional Testing tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

Syntax

You may also assign a timeout value using the `TransactionTime` property.

```
wlGlobals.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:

```
wlGlobals.TransactionTime = 1
```

-Or-

```
wlGlobals.TransactionTime = "0.001"
```

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- *Transaction Verification Components* (on page 36)
- VerificationFunction() (user-defined) (see *VerificationFunction() (user-defined) (function)* on page 297)

type (property)

Property of Objects

- element (see *element (object)* on page 80)
- form (see *form (object)* on page 95)
- wlHttp (see *wlHttp (object)* on page 316)
- wlHttp.Data (see *Data (property)* on page 66)
- wlHttp.DataFile (see *DataFile (property)* on page 67)

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`".

Note: 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.

Syntax

element:

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

```
<NA>
```

wlHttp.Data:

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

```
wlHttp.Data.Type = "application/x-www-form-urlencoded"
```

Comment

The `Type` property for `wlHttp.Data` and `wlHttp.DataFile` objects is written in uppercase.

See also

- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Post() (see *Post() (method)* on page 205)
- value (see *value (property)* on page 294)
- wlClear() (see *wlClear() (method)* on page 301)
- wlHttp (see *wlHttp (object)* on page 316)

Url (property)

Property of Objects

- element (see *element (object)* on page 80)
- form (see *form (object)* on page 95)
- frames (see *frames (object)* on page 99)
- Image (see *Image (object)* on page 149)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

- wlHttp (see *wlHttp (object)* on page 316)

Description

Sets or retrieves the URL of the parent object on the Web page (read-only). If the parent object is of type <IMG>, 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 `wlMetas`, 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"
```

wlMetas:

When working with `wlMetas` objects, use the all-uppercase caps form:

```
CurrentLink = document.wlMetas[0].URL
```

Comment

The `URL` property for `area`, `element`, `form`, `frame`, `image`, `link`, `location`, and `wlMetas` objects is written in all-uppercase caps.

See also

- *HTTP Components* (on page 24)
- content (see *content (property)* on page 56)
- httpEquiv (see *httpEquiv (property)* on page 144)
- Name (see *Name (property)* on page 174)
- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)
- wlMetas (see *wlMetas (object)* on page 320)

UserAgent (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

A user agent is the client application used with a particular network protocol. The phrase, "user agent" is most commonly used in reference to applications that access the World Wide Web. Web user agents range from web browsers to search engine crawlers ("spiders"), as well as mobile phones, etc. The user agent string can be sent as part of the HTTP request, prefixed with User-agent: or User-Agent:. This string typically includes information such as the application name, version, host, host operating system, and language. Some examples of user agent strings can be found at:

http://en.wikipedia.org/wiki/User_agent#Example_user-agent_strings

The UserAgent property is used to define the user agent string for the scope of the WebLOAD object with which it is associated.

GUI mode

WebLOAD recommends setting user agent values through the WebLOAD Console. Select a browser type and user agent through the Browser Parameters tab of the **Default** or **Current Project Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)

UserName (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The user name that the script 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 WebLOAD Console. Enter user authentication information through the Authentication tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

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"
```

Comments

A user is only authenticated once during a round with a set of credentials. Each subsequent request will use these credentials regardless of what is contained in the script. If the value of these credentials are changed after authentication, they will only be used during the next round, not during the current round.

For example, if you are trying to send a request to a URL with a group of users (user1, user2, and user3), but user1 has already been authenticated, the login is always performed for user1 until the round is complete.

See also

- *HTTP Components* (on page 24)
- NTUserName, NTPassWord (see *NTUserName, NTPassWord (properties)* on page 176)

UseSameProxyForSSL (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

The UseSameProxyForSSL property can have one of the following values:

- **false** (default value) – The engine uses the Proxy, ProxyUserName, ProxyPassWord, ProxyNTUserName, and ProxyNTPassWord properties for both SSL and non-SSL traffic.
- **true** – The engine uses the Proxy, ProxyUserName, ProxyPassWord, ProxyNTUserName, and ProxyNTPassWord properties for non-SSL traffic and the HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord, HttpsProxyNTUserName, and HttpsNTPassWord properties for SSL traffic.

This property is used when you are working with a separate SSL proxy.

Note: This property can only be inserted manually.

Example

```
wlGlobals.UseSameProxyForSSL = false
```

See also

- *HTTP Components* (on page 24)
- *Security* in the *WebLOAD Scripting Guide*
- HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (see *HttpsProxy, HttpsProxyUserName, HttpsProxyPassWord (properties)* on page 145)
- HttpsProxyNTUserName, HttpsProxyNTPassWord (see *HttpsProxyNTUserName, HttpsProxyNTPassWord (properties)* on page 146)

UsingTimer (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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

- *HTTP Components* (on page 24)
- SendTimer() (see *SendTimer() (function)* on page 239)
- SetTimer() (see *SetTimer() (function)* on page 244)

value (property)

Property of Objects

- element (see *element (object)* on page 80)
- option (see *option (object)* on page 185)
- wlHeaders (see *wlHeaders (object)* on page 314)
- wlHttp.Data (see *Data (property)* on page 66)
- wlHttp.Header (see *Header (property)* on page 140)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

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 `wlHeaders` or `wlSearchPairs` 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

The `Value` property for `element` and `wlHttp.Data` objects is written in uppercase.

See also

- *Collections* (on page 27)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- element (see *element (object)* on page 80)
- Erase (see *Erase (property)* on page 88)
- FileName (see *FileName (property)* on page 93)
- form (see *form (object)* on page 95)
- FormData (see *FormData (property)* on page 97)
- Get() (see *Get() (transaction method)* on page 104)
- Header (see *Header (property)* on page 140)
- Image (see *Image (object)* on page 149)
- key (see *key (property)* on page 160)
- option (see *option (object)* on page 185)
- Post() (see *Post() (method)* on page 205)
- *Select* (on page 230)

- type (see *type (property)* on page 288)
- value (see *value (property)* on page 294)
- wlClear() (see *wlClear() (method)* on page 301)
- wlHeaders (see *wlHeaders (object)* on page 314)
- wlHttp (see *wlHttp (object)* on page 316)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

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 (see *ClientNum (variable)* on page 50), which provides an identification number that is only unique within a single Load Generator. The identification string is composed of a concatenation of the script name, Load Generator name, current thread number, and round number.

Syntax

```
VCUniqueID()
```

Return Value

Returns a unique identification string for the current Virtual Client instance.

Example

```
InfoMessage(VCUniqueID())
```

The results are

```
j@chaimsh.0.1
```

where:

- `j` is the name of the script.
- `chaimsh` is the name of the Load Generator.
- `0` is the client number.
- `1` is the round number.

See also

- ClientNum (see *ClientNum (variable)* on page 50)
- GeneratorName() (see *GeneratorName() (function)* on page 101)

- GetOperatingSystem() (see *GetOperatingSystem() (function)* on page 131)
- *Identification Variables and Functions* (on page 29)
- RoundNum (see *RoundNum (variable)* on page 222)

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 script 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 Log Window, similar to any other WebLOAD test failure. Refer to the *WebLOAD Console 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 script 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 script displays an error message in the Log window. If you are working with

WebLOAD, a new round is begun only if WebLOAD is configured for multiple iterations.

- `WLSevereError` – This specific transaction failed and the test session must be stopped completely. The script displays an error message in the Log window. If you are working with WebLOAD Recorder, the test session is stopped. If you are working with WebLOAD, 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, the default assumption is 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

All functions must be declared in the script before they can be called.

For a more complete explanation and examples of functional testing and transaction verification, see the *WebLOAD Scripting Guide*.

See also

- BeginTransaction() (see *BeginTransaction() (function)* on page 42)
- CreateDOM() (see *CreateDOM() (function)* on page 63)
- CreateTable() (see *CreateTable() (function)* on page 65)
- EndTransaction() (see *EndTransaction() (function)* on page 88)
- ReportEvent() (see *ReportEvent() (function)* on page 218)
- SetFailureReason() (see *SetFailureReason() (function)* on page 243)
- TimeoutSeverity (see *TimeoutSeverity (property)* on page 283)
- TransactionTime (see *TransactionTime (property)* on page 287)
- *Transaction Verification Components* (on page 36)

Version (property)

Property of Objects

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)
- wlLocals (see *wlLocals (object)* on page 319)

Description

Stores the HTTP version number for the current test session. Current supported versions include 1.0 and 1.1.

GUI mode

WebLOAD recommends selecting an HTTP version through the WebLOAD Console. Click the appropriate version number radio button in the HTTP Parameters tab of the **Default** or **Current Session Options** dialog box, accessed from the **Tools** tab of the ribbon.

See also

- *HTTP Components* (on page 24)

WarningMessage() (function)

Description

Use this function to display a warning message in the Log window.

Syntax

```
WarningMessage(msg)
```

Parameters

Parameter Name	Description
`msg`	A string with a warning message to be sent to the Log window.

Comment

If you call `WarningMessage()` in the main script, WebLOAD sends a warning message to the Log window and continues with script 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 script files directly through the WebLOAD Recorder. Drag the Message icon from the WebLOAD Recorder toolbox

into the script. The Message dialog box opens. Enter the message text, select the WLMinorError severity level for the message, and click **OK**.

See also

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)
- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlException (see *wlException (object)* on page 306)
- wlException() (see *wlException() (constructor)* on page 308)

window (object)

Property of Object

- frames (see *frames (object)* on page 99)

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: 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
```

Properties

- document (see *document (object)* on page 78)
- location (see *location (object)* on page 168)
- Name (see *Name (property)* on page 174)
- title (see *title (property)* on page 355)
- Url (see *Url (property)* on page 363)

See also

- *Collections* (on page 27)

wlClear() (method)

Method of Objects

The `wlHttp` object includes the following collections for storing data. These data storage collections each include the method `wlClear()`.

- wlHttp.Data (see *Data (property)* on page 66)
- wlHttp.DataFile (see *DataFile (property)* on page 67)
- wlHttp.FormData (see *FormData (property)* on page 97)
- wlHttp.Header (see *Header (property)* on page 140)

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.

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* (on page 27)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- FormData (see *FormData (property)* on page 97)
- Header (see *Header (property)* on page 140)
- wlHttp (see *wlHttp (object)* on page 316)

wlCookie (object)

Description

The `wlCookie` object gets, sets and deletes cookies. These activities may be required by an HTTP server.

Note: 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 a script 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 a script. 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.
- A thread can get the value of a cookie that is 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.

Note: This property can only be inserted manually.

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")
//Get the value of a cookie
retValue = wlCookie.Get("CUSTOMER", "www.ABCDEF.com", "/" )
//Delete the cookie
wlCookie.ClearAll()
```

Methods

- ClearAll() (see *ClearAll() (method)* on page 48)
- Delete() (see *Delete() (cookie method)* on page 75)
- Set() (see *Set() (cookie method)* on page 241)
- Get() (see *Get() (cookie method)* on page 103)

wlDataFileField (method)

Description

`WLDataFileField` creates the data file field parameter.

Syntax

```
fileFieldParam = WLDataFileField(paramName, ColumnNumber);
```

Parameters

Parameter Name	Description
paramName	File parameter ID, returned by `WLDataFileParam`.
ColumnNumber	File column number.

wlDataFileParam() (parameterization)

Description

Define a data file parameter.

Syntax

```
<paramName> = wlDataFileParam(FileID, CopyFileId, HeaderLines,
Delimiter, AccessMethod, Scope, UsageMethod, EndOfFileBehavior);
```

Parameters

Parameter Name	Description
FileID	A string which is a unique parameter identifier.
CopyFileId	An identifier which refers to the local file. This value is returned by the CopyFile command.
HeaderLines	A parameter that defines the number of header lines the file contains. All values are enumerated numeric values. Possible values are: • `0`. The file does not contain any header lines. This is the default value. • `<X>`. Where <X> is any number above zero. The file contains <X> header lines at the beginning of the file. The values contained in these header lines are not used as parameters but as variable names in the JavaScript code.
Delimiter	Character used to separate fields in one line of the input file. The default delimiter character is a comma.

Parameter Name	Description
AccessMethod	Defines the method for reading the next row from the file. All values are enumerated numeric values. Possible values are: • `WLParamRandom`. Gets a random row from the file. • `WLParamOrdered`. Every client gets the next row from the file (order is important). • `WLParamNotOrdered`. Every client gets the next row from the file (order is not important).
Scope	Defines the scope (sharing policy) of the parameter. Possible values are: • `WLParamGlobal`. All virtual clients read rows from the shared (global) pool. • `WLParamLocal`. Each virtual client reads rows from its own copy of the pool. • `WLParamGlobalLocked`. All virtual clients read a unique row from the global pool, which is shared by all virtual clients on all load generators.
UsageMethod	Defines when the parameter is updated, meaning when a new value will be read. Possible values are: • `WLParamUpdateRound`. The script reads a new row from the file one time for each round. Using the same parameter again in the same round will result in the same value. • `WLParamUpdateOnce`. The script reads a new row from the file once at the beginning of the test (in `InitClient`). Every usage of the parameter by that Virtual Client will always result in the same value. • `WLParamUpdateUse`. The parameter's value will be read each time it is used.
EndOfFileBehavior	Defines how WebLOAD behaves when it reaches the end of the file. All values are enumerated numeric values. • `WLParamKeepLast`. Keep the last value. • `WLParamCycle`. Start from the beginning of the file. Each row can be used any number of times. • `WLParamStopVC`. Abort the specific Virtual Client that tried to read past the end of the file. An error message is written to the log file.

Example

```
function InitAgenda()
{
myFileParam_File = CopyFile("C:\\My
Documents\\WebLOAD\\Sessions\\param1.txt")
}
function InitClient()
```

```
{
myFileParam_DataFileParam = wlDataFileParam (
"myFileParam",myFileParam_File,
1,",",wlParamRandom,WLParamGlobalLocked,wlParamUpdateRound,wlParamCycl
e);
myFileParam_col1 = wlDataFileField( myFileParam_DataFileParam, 1);


myFileParam_col2 = wlDataFileField( myFileParam_DataFileParam, 2);
}
/***** WLIDE - Message - ID:4 *****/
InfoMessage(myFileParam_col1.getValue())

 // END WLIDE

/***** WLIDE - Message - ID:5 *****/
InfoMessage(myFileParam_col2.getValue())
```

Methods

- wlDataFileField() (see *wlDataFileField (method)*on page 304)

wlException (object)

Description

script 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 scripts 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.

Users may add their own `try()/catch()` pairs to a script, 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. If an error is caught while the script is in the middle of executing the code within a protected logical code block (by `try()`),

WebLOAD will detour to a user-defined error function (the `catch()` block) and then continue execution with the next navigation block in the script.

`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:

```
try{
  ...
  //do a lot of things
  ...
  //error occurs here
  ...
}

catch(e){
  myException = new wlException(e,"we have a problem")
  //things to do in case of error
  if (myException.GetSeverity() == WLError) {
     // Do one set of Error activities
     myException.ReportLog()
     throw myException
  }
  else {
     // Do a different set of Severe Error activities
     throw myException
  }
}
```

Methods

- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- ReportLog() (see *ReportLog() (method)* on page 219)
- wlException() (see *wlException() (constructor)* on page 308)

See also

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- InfoMessage() (see *InfoMessage() (function)* on page 153)

- *Message Functions* (on page 30)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)

wlException() (constructor)

Method of Object

- wlException (see *wlException (object)* on page 306)

Description

Creates a new `wlException` object.

Syntax

```
NewExceptionObject = new wlException(severity, message)
```

Parameters

Parameter Name	Description
`severity`	One of the following integer constants: • `WLError`. This specific transaction failed and the current test round was aborted. The script 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 script 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

- ErrorMessage() (see *ErrorMessage() (function)* on page 90)
- GetMessage() (see *GetMessage() (method)* on page 129)
- GetSeverity() (see *GetSeverity() (method)* on page 134)
- InfoMessage() (see *InfoMessage() (function)* on page 153)
- *Message Functions* (on page 30)

- ReportLog() (see *ReportLog() (method)* on page 219)
- SevereErrorMessage() (see *SevereErrorMessage() (function)* on page 246)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- WarningMessage() (see *WarningMessage() (function)* on page 299)

wlGeneratorGlobal (object)

Description

WebLOAD provides a global object called `wlGeneratorGlobal`. The `wlGeneratorGlobal` object enables sharing of global variables and values between all threads of a single Load Generator, even when running multiple scripts. (Compare to the wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)object, 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 wlGlobals (see *wlGlobals (object)* on page 313) object, which enables sharing of global variables and values between threads of a single script, 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 `wlGeneratorGlobal` object for each Load Generator participating in a test session. Use the `wlGeneratorGlobal` methods to create and access variable values that you wish to share between threads of a Load Generator. Edit `wlGeneratorGlobal` properties and methods through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor* (on page 18). While global variables may be accessed anywhere in your script, be sure to initially declare `wlGeneratorGlobal` values in the `InitAgenda()` *function only*. Do not define new values within the main body of a Script, for they will not be shared correctly by all threads.

Methods

The `wlGeneratorGlobal` object includes the following methods:

- Add() (see *Add() (method)* on page 39)
- Get() (see *Get() (addition method)* on page 102)
- Set() (see *Set() (addition method)* on page 240)

Properties

`wlGeneratorGlobal` incorporates a dynamic property set that consists of whatever global variables have been defined, set, and accessed by the user through the `wlGeneratorGlobal` method set only.

See also

- wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)

wlGet() (method)

Method of Object

Each of the different types of collections of elements found in the parsed DOM tree includes 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

Parameter Name	Description
`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 website included a form with a field called `length`. This could lead to a confusing situation, where the word `length` appearing in a script could represent either the number of elements in a `FormData` array, as explained in length, 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 `wlGet()` to retrieve field data whenever the name could lead to potential ambiguity. When recording scripts with WebLOAD Recorder, WebLOAD recognizes potential ambiguities and inserts the appropriate `wlGet()` statements automatically.

See also

- *Collections* (on page 27)
- wlHttp (see *wlHttp (object)* on page 316)

wlGetAllForms() (method)

Method of Object

- document (see *document (object)* on page 78)

Description

Retrieve a collection of all forms (<FORM> elements) in an HTML page and its nested frames.

Syntax

```
wlGetAllForms()
```

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.

See also

- *HTTP Components* (on page 24)

wlGetAllFrames() (method)

Method of Object

- document (see *document (object)* on page 78)

Description

Retrieve a collection of all frames in an HTML page, at any level of nesting.

Syntax

```
wlGetAllFrames()
```

Return Value

A collection that includes the top-level frame (from which you called the method) and all its subframes.

See also

- *HTTP Components* (on page 24)

wlGetAllLinks() (method)

Method of Object

- document (see *document (object)* on page 78)

Description

Retrieve a collection of all links (<A> elements) in an HTML page and its nested frames.

Syntax

```
wlGetAllLinks()
```

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.

See also

- *HTTP Components* (on page 24)

wlGlobals (object)

Description

The `wlGlobals` object stores the default global configuration properties set by the user through the WebLOAD Recorder or Console, including properties defining expected dialog boxes, verification test selections, and dynamic state management.

`wlGlobals` is a global object, whose property values are shared by all threads of a script running on a single Load Generator. The `wlGlobals` object enables sharing of user-defined global variables and values between threads of a single script, running on a single Load Generator. (Compare to the wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309) object, which enables sharing of global variables and values between all threads of a single Load Generator, and the wlSystemGlobal (see *wlSystemGlobal (object)* on page 332) object, which enables sharing of global variables and values system-wide, between all threads of all Load Generators participating in a test session.)

Note: Most global configuration property values and user-defined variables should be set through the WebLOAD Recorder or Console. The property descriptions here are intended mainly to explain the lines of code seen in the JavaScript View of the WebLOAD Recorder desktop. Syntax details are also provided for the benefit of users who prefer to manually edit the JavaScript code of their scripts through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor* (on page 18). If you do decide to edit the global variable values in your script, set `wlGlobals` properties in the `InitAgenda()` *function only*. Do not define new values within the main body of a script. The values will not be shared correctly by all script threads.

The configuration properties of the `wlGlobals` object are almost all duplicated in the wlLocals (see *wlLocals (object)* on page 319), which contains the local configuration settings for browser actions, and in the wlHttp (see *wlHttp (object)* on page 316), 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 the *WebLOAD Scripting Guide*.

Properties

The `wlGlobals` object includes the following property classes:

- *Automatic State Management for HTTP Protocol Mode* (on page 24)
- *HTTP Components* (on page 24)
- *Transaction Verification Components* (on page 36)

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 script files.

The recommended way to set configuration values is through the WebLOAD Recorder, using the Default, Current, and Global Options dialog boxes accessed from the **Tools** tab in the Console desktop ribbon. The dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

- wlHttp (see wlHttp (object) on page 316)
- wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309)
- wlLocals (see *wlLocals (object)* on page 319)
- wlSystemGlobal (see *wlSystemGlobal (object)* on page 332)

wlHeaders (object)

Property of Objects

Headers on a Web page are accessed through `wlHeaders` objects that are grouped into collections of `wlHeaders`. The `wlHeaders` collection is a property of the following objects:

- document (see *document (object)* on page 78)

Description

Each `wlHeaders` object contains a key-value pair. `wlHeaders` 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 *wlSearchPairs (object)* (on page 327).)

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

Syntax

`wlHeaders` objects are grouped together within collections of `wlHeaders`. To access an individual `wlHeaders`'s properties, check the `length` property of the `wlHeaders` collection and use an index number to access the individual `wlHeaders` object, with the following syntax:

```
NumberofHeaderObjects = document.wlHeaders.length
document.wlHeaders[index#].<wlHeaders-property>
```

Example

WebLOAD stores the header pairs from the response to 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 `wlHeaders` object includes the following properties:

- key (see *key (property)* on page 160)
- value (see *value (property)* on page 294)

See also

- *Collections* (on page 27)
- Header (see *Header (property)* on page 140)
- wlSearchPairs (see *wlSearchPairs (object)* on page 327)

wlHtml (object)

Description

If your script 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 a script. You cannot manually declare `wlHtml` objects yourself.

Methods

- GetFieldValue() (see *GetFieldValue() (method)* on page 113)
- GetFieldValueInForm() (see *GetFieldValueInForm() (method)* on page 114)
- GetFormAction() (see *GetFormAction() (method)* on page 115)
- GetFrameByUrl() (see *GetFrameByUrl() (method)* on page 116)
- GetFrameUrl() (see *GetFrameUrl() (method)* on page 117)
- GetHeaderValue() (see *GetHeaderValue() (method)* on page 118)
- GetHost() (see *GetHost() (method)* on page 119)
- GetHostName() (see *GetHostName() (method)* on page 120)
- GetLinkByName() (see *GetLinkByName() (method)* on page 127)
- GetLinkByUrl() (see *GetLinkByUrl() (method)* on page 128)
- GetPortNum() (see *GetPortNum() (method)* on page 132)
- GetQSFieldValue() (see *GetQSFieldValue() (method)* on page 133)
- GetStatusLine() (see *GetStatusLine() (method)* on page 135)
- GetStatusNumber() (see *GetStatusNumber() (method)* on page 136)
- GetUri() (see *GetUri() (method)* on page 137)

wlHttp (object)

Description

The `wlHttp` 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 wlGlobals (see *wlGlobals (object)* on page 313), which contains the default global configuration settings for browser actions, and in the wlLocals (see *wlLocals (object)*on page 319), 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 the *WebLOAD Scripting Guide*. The `wlHttp` object also contains the methods that implement the user activities saved during the WebLOAD Recorder 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 `wlHttp` object.

Properties and Methods

The `wlHttp` object includes the following property and method classes:

- *Automatic State Management for HTTP Protocol Mode* (on page 24)
- *HTTP Components* (on page 24)
- *Transaction Verification Components* (on page 36)

Syntax

Each individual function class includes the syntax specifications that apply to that class.

GUI mode

The `wlHttp` property and method descriptions explain how to explicitly set values for these session configuration properties within your JavaScript script files.

Note: The recommended way to set configuration values is through the WebLOAD Recorder, using the Default, Current, and Global Options dialog boxes accessed from the **Tools** tab in the Console desktop ribbon. The dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlLocals (see *wlLocals (object)* on page 319)

wlInputFile (object)

Description

The `wlInputFile` object supports reading values from an external text file. This is useful when you need to parameterize your script. The input file supports the following access methods:

- Unique access to a parameters file's record. This ensures that a value that was read by VC1 will not be read by any other VC as long as VC1 is using this value.
- Shared access for a parameters file among Load Generators and Load Machines and among different scripts.
- Sequential and random access to a parameters file.

The `wlInputFile` object enables Load Generators running on more than one load machine to access a single parameters file in a way that enables unique reading of the parameters from the file. In addition, a single parameters file can be accessed by more

than one script in a way that enables unique reading of parameters from the file by all of the scripts.

Create `wlInputFile` objects and manage your files using the constructor and methods described in this section.

Syntax

```
MyFileObj = new wlInputFile(fileID)
```

Parameters

Parameter Name	Description
`fileID`	An identifier which refers to the local file. This value is returned by the CopyFile command.

Example

```
fileID = CopyFile(<full path>)
…
MyFileObj = new wlInputFile(fileID)
…
MyFileObj.Open([AccessMethod], [ShareMethod], [UsageMethod],
[EndOfFileBehavior], [HeaderLines])
```

Methods

- Open() (see *Open() (method)* on page 180)
- GetLine() (see *GetLine() (function)* on page 123)

See also

- *Using the IntelliSense JavaScript Editor* (on page 18)
- CopyFile() (see *CopyFile() (function)* on page 61)

wlInputFile() (constructor)

Method of Object

- wlInputFile (see *wlInputFile (object)* on page 317)

Description

Creates a new `wlInputFile` object. For optimal performance, construct a new file object in the `InitClient` section of your script. The file is copied from the Console to the Load Generator. The input file specified in the `wlInputFile` object is opened.

Syntax

```
myFileObj = new wlInputFile(fileID)
```

Parameters

Parameter Name	Description
`fileID`	An identifier which refers to the local file. This value is returned by the CopyFile command.

Return Value

A pointer to a new `wlInputFile` object.

Example

```
fileID = CopyFile(<full path>)
…
MyFileObj = new wlInputFile(fileID)
…
MyFileObj.Open([AccessMethod], [ShareMethod], [UsageMethod],
[EndOfFileBehavior], [HeaderLines])
```

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 a script. You cannot declare `wlLocals` objects yourself.

The properties of the `wlLocals` object are all duplicated in the wlGlobals (see *wlGlobals (object)* on page 313), which contains the default global settings, and in the wlHttp (see *wlHttp (object)* on page 316), which contains the settings for an immediate action. To understand how there could potentially be three different settings for a single configuration property, see the *WebLOAD Scripting Guide*.

Properties

The `wlLocals` object includes the following property classes:

- *Automatic State Management for HTTP Protocol Mode* (on page 24)
- *HTTP Components* (on page 24)
- *Transaction Verification Components* (on page 36)

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 script files.

Note: The recommended way to set configuration values is through the WebLOAD Recorder, using the Default, Current, and Global Options dialog boxes accessed from the **Tools** tab in the Console desktop ribbon. The dialog boxes provide a means of defining and setting configuration values with ease, simplicity, and clarity.

See also

- wlGlobals (see *wlGlobals (object)* on page 313)
- wlHttp (see *wlHttp (object)* on page 316)

wlMetas (object)

Property of Objects

META objects on a Web page are accessed through `wlMetas` objects that are grouped into collections of `wlMetas`. The `wlMetas` collection is a property of the following objects:

- document (see *document (object)* on page 78)

Description

Each `wlMetas` object stores the parsed data for an HTML meta object (<META> tag). `wlMetas` objects are local to a single thread. You cannot create new `wlMetas` objects using the JavaScript `new` operator, but you can access them through the properties and methods of the standard DOM objects. `wlMetas` properties are read only.

Syntax

`wlMetas` objects are grouped together within collections of `wlMetas`. To access an individual `wlMetas`'s properties, check the length property of the `wlMetas` collection and use an index number to access the individual `wlMetas` object, with the following syntax:

```
NumberofMetaObjects = document.wlMetas.length
document.wlMetas[#].<wlMetas-property>
```

Example

To find out how many `wlMetas` objects are contained within a document header, check the value of:

```
document.wlMetas.length
```

Access each `wlMetas`'s properties directly using the preceding syntax. For example:

```
document.wlMetas[0].key
```

Properties

The `wlMetas` object includes the following properties:

- content (see *content (property)* on page 56)
- httpEquiv (see *httpEquiv (property)* on page 144)
- Name (see *Name (property)* on page 172)
- Url (see *Url (property)* on page 289)

See also

- *Collections* (on page 27)

wlNumberParam() (parameterization)

Description

Define a number parameter.

Syntax

```
paramName = wlNumberParam (ParamID, MinValue, MaxValue,Step,
AccessMethod, Scope, UsageMethod, OutOfValuesBehavior);
```

Parameters

Parameter Name	Description
ParamID	A string that is a unique parameter identifier.
MinValue	The minimum value of the number range.
MaxValue	The maximum value of the number range.
Step	The increment between numbers.
AccessMethod	Defines the method for calculating the next value from the range. All values are enumerated numeric values. Possible values are: • `wlParamRandom`. Gets a random value from the range. • `wlParamOrdered`. Every client gets the next value from the range (order is important). • `wlParamNotOrdered`. Every client gets the next value from the range (order is not important).

Parameter Name	Description
Scope	Defines the scope (sharing policy) of the parameter. Possible values are: • `wlParamGlobal`. All virtual clients read values from the shared (global) pool. • `wlParamLocal`. Each virtual client reads values from its own pool. • `wlParamGlobalLocked`. All virtual clients read unique values from the shared (global) pool.
Usage Method	Defines when the parameter is updated, meaning when a new value will be read. Possible values are: • `WLParamUpdateRound`. The script reads a new value from the file once for each round. Using the same parameter again in the same round will result in the same value. • `WLParamUpdateOnce`. The script reads a new value from the file once at the beginning of the test (in `InitClient`). All usage of the parameter by that Virtual Client will always result in the same value. • `WLParamUpdateUse`. The parameter's value will be read each time it is used.
OutOfValuesBehavior	Defines how WebLOAD behaves when it reaches the end of the range. All values are enumerated numeric values. Possible values are: • `WLParamKeepLast`. Keep the last value. • `WLParamCycle`. Start from the beginning of the range. • `WLParamStopVC`. Abort the specific Virtual Client that tried to read past the end of the range. An error message is written to the log file.

Example

```
function InitClient()
{
NewParam1 = wlNumberParam("NewParam1",1, 100, 1, wlParamRandom,
wlParamLocal, wlParamUpdateRound, wlParamCycle);
}
/***** WLIDE - Message - ID:3 *****/
InfoMessage(NewParam1.getValue())

 // END WLIDE
```

wlOutputFile (object)

Description

The `wlOutputFile` object writes script output messages to a global output file. Create `wlOutputFile` objects and manage your files using the constructor and methods described in this section.

Syntax

```
MyFileObj = new wlOutputFile("filename")
…
MyFileObj.Write("Happy Birthday")
…
delete MyFileObj
```

Example

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

Methods

- Close() (see *Close() (function)* on page 52)
- remove() (see *remove() (method)* on page 74)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- wlOutputFile() (see *wlOutputFile() (constructor)* on page 324)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

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.

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 script data.

If you declare a new `wlOutputFile` object in the `InitAgenda()` function of a script, the output file will be shared by all the script 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 script execution.

If you declare a new `wlOutputFile` object in the `InitClient()` function or main body of a script, 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 a script, and then run your script 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 a script, 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() (see *CopyFile() (function)* on page 61)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- *Using the IntelliSense JavaScript Editor* (on page 18)

wlOutputFile() (constructor)

Method of Object

- wlOutputFile (see *wlOutputFile (object)* on page 323)

Description

To create a new `wlOutputFile` object, use the `wlOutputFile()` constructor.

Syntax

```
new wlOutputFile(filename)
```

Parameters

Parameter Name	Description
`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 script data.

If you declare a new `wlOutputFile` object in the `InitAgenda()` function of a script, the output file will be shared by all the script 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 script execution.

If you declare a new `wlOutputFile` object in the `InitClient()` function or main body of a script, 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 a script, and then run your script 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 a script, 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

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)

- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile (see *wlOutputFile (object)* on page 323)
- Write() (see *Write() (method)* on page 343)
- Writeln() (see *Writeln() (method)* on page 344)

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 script. 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)
```

Methods

- Num() (see *Num() (method)* on page 177)
- Range() (see *Range() (method)* on page 215)
- Seed() (see *Seed() (method)* on page 229)
- Select() (see *Select() (method)* on page 230)

wlSearchPairs (object)

Method of Object

- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)

Description

Each `wlSearchPairs` 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 wlHeaders (see *wlHeaders (object)* on page 314). Information found in *request headers* is available through the `wlHttp.Header` (see *Header (property)* on page 140) property.)

`wlSearchPairs` objects are grouped into `wlSearchPairs` collections, where the collections are themselves properties of the `link` and `location` objects.

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

Syntax

`wlSearchPairs` objects are grouped together within collections of `wlSearchPairs`. To access an individual `wlSearchPairs`'s properties, check the `length` property of the `wlSearchPairs` collection and use an index number to access the individual `wlSearchPairs` object, with the following syntax:

```
NumberofSearchPairObjects =
                        document.links[1].wlSearchPairs.length
document.links[1].wlSearchPairs[index#].<wlSearchPairs-property>
```

Example

To find out how many `wlSearchPairs` objects are contained within a document's link, check the value of:

```
document.links[1].wlSearchPairs.length
```

Access each `wlSearchPairs`'s properties directly through the index number of that item. For example:

```
document.links[1].wlSearchPairs[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 script:

```
function InitAgenda() {
  wlGlobals.Url = "http://www.ABCDEF.com"
  //Enable link parsing
  wlGlobals.ParseLinks = true
}
wlHttp.Get()
```

For the link in question, WebLOAD stores the attribute pairs in the `document.links[2].wlSearchPairs` property. This property is actually a collection containing two `wlSearchPairs` objects. The following is a complete listing of the collection.

```
document.links[2].wlSearchPairs[0].key = "Product"
document.links[2].wlSearchPairs[0].value = "modems"
document.links[2].wlSearchPairs[1].key = "Type"
document.links[2].wlSearchPairs[1].value = "ISDN"
```

Properties

The `wlSearchPairs` object includes the following properties:

- key (see *key (property)* on page 160)
- value (see *value (property)* on page 294)

See also

- *Collections* (on page 27)
- Header (see *Header (property)* on page 140)
- link (see *link (object)* on page 162)
- location (see *location (object)* on page 168)
- wlHeaders (see *wlHeaders (object)* on page 314)
- wlHttp (see *wlHttp (object)* on page 316)

wlSet() (method)

Method of Objects

The `wlHttp` object includes the following collections for storing data. These data storage collections each include the method `wlSet()`.

- wlHttp.Data (see *Data (property)* on page 66)
- wlHttp.DataFile (see *DataFile (property)* on page 67)
- wlHttp.FormData (see *FormData (property)* on page 97)

- wlHttp.Header (see *Header (property)* on page 140)

Description

`wlSet()` 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

```
wlHttp.Collection.wlSet(FieldName, Value)
```

Parameters

Parameter Name	Description
`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

```
wlHttp.FormData.wlSet("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 website included a form with a field called `length`. This could lead to a confusing situation, where the word `length` appearing in a script could represent either the number of elements in a `FormData` array, as explained in length, 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 scripts 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* (on page 27)
- Data (see *Data (property)* on page 66)
- DataFile (see *DataFile (property)* on page 67)
- FormData (see *FormData (property)* on page 97)
- Header (see *Header (property)* on page 140)
- wlHttp (see *wlHttp (object)* on page 316)

wlSource (property)

Property of Object

- document (see *document (object)* on page 78)

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, you must enable the SaveSource (see *SaveSource (property)* on page 226) property of the `wlGlobals`, `wlLocals`, or `wlHttp` object. To save the source in a file, use the Outfile property (see *Outfile (property)* on page 188).

See also

- Outfile (see *Outfile (property)* on page 188)
- SaveSource (see *SaveSource (property)* on page 226)

wlStatusLine (property)

Property of Object

- document (see *document (object)* on page 78)

Description

The status line of the HTTP header (read-only string, for example “OK”).

Syntax

```
document.wlHeaders[“status line”]
```

wlStatusNumber (property)

Property of Object

- document (see *document (object)* on page 78)

Description

The HTTP status value, which WebLOAD retrieves from the HTTP header (read-only integer, for example 200).

Syntax

```
document.wlStatusNumber
```

wlStringParam() (parameterization)

Description

Define a random string parameter.

Syntax

```
<varName> = wlStringParam(minLength, maxLength, usage)
```

Parameters

Parameter Name	Description
minLength	The minimum string length (number of characters),
maxLength	The maximum string length (number of characters).

`usage`	Defines when the parameter is updated, meaning when a new value will be calculated. Possible values are: • `wlParamUpdateRound`. The parameters value will be calculated once for each round. Using the same parameter again in the same round will result with the same value. • `wlParamUpdateOnce`. The parameter's value will be calculated once per each Virtual Client (in its `InitClient` function). All usage of the parameter by that Virtual Client will always result in the same value. • `wlParamUpdateUse`. The parameter's value will be calculated each time it is used.

Example

```
function InitClient()
{
NewParam1 = wlStringParam(2, 7, wlParamUpdateUse);
}
/***** WLIDE - Message - ID:3 *****/
InfoMessage(NewParam1.getValue())

 // END WLIDE
```

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 scripts running on multiple Load Generators. (Compare to the wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309), which enables sharing of global variables and values between all threads of a single Load Generator, and to the wlGlobals (see *wlGlobals (object)* on page 313), which enables sharing of global variables and values between all threads of a single script, 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* (on page 18). While global variables may be accessed anywhere in your script, be sure to initially declare `wlSystemGlobal` values in the `InitAgenda()` *function only*. Do not define

new values within the main body of a script, for they will not be shared correctly by all threads.

Methods

- Add() (see *Add() (method)* on page 39)
- Get() (see *Get() (addition method)* on page 102)
- Set() (see *Set() (addition method)* on page 240)

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

- wlGeneratorGlobal (see *wlGeneratorGlobal (object)* on page 309)

wlTables (object)

Property of Object

TABLE objects on a Web page are accessed through `wlTables` objects that are grouped into collections of `wlTables`. The `wlTables` collection is a property of the following object:

- document (see *document (object)* on page 78)

Description

Each `wlTables` 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 `wlTables` object is also linked to `row` and `cell` objects and their properties.

`wlTables` 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 `wlTables`'s properties, check the `length` property of the `wlTables` collection and use an index number to access the individual `wlTables` object, with the following syntax:

```
NumberofTableObjects = document.wlTables.length
document.wlTables[index#].<wlTables-property>
```

Example

Access each `wlTables`'s properties directly through the index number of that item. For example:

```
document.wlTables[0].cols
```

`wlTables` objects may also be accessed directly using the table ID. This is illustrated in the *id* property description.

Properties

Each `wlTables` object contains information about the data found in the whole table, organized by rows, columns, and cells. The `wlTables` object includes the following properties:

- cell (see *cell (object)* on page 44) (wlTables and row property)
- cols (see *cols (property)* on page 54) (wlTables property)
- id (see *id (property)* on page 146) (wlTables property)
- row (see *row (object)* on page 223) (wlTables property)

See also

- cellIndex (see *cellIndex (property)* on page 46) (cell property)
- *Collections* (on page 27)
- Compare() (see *Compare() (method)* on page 55)
- CompareColumns (see *CompareColumns (property)* on page 55)
- CompareRows (see *CompareRows (property)* on page 55)
- Details (see *Details (property)* on page 76)
- InnerHTML (see *InnerHTML (property)* on page 154) (cell property)
- InnerText (see *InnerText (property)* on page 156) (cell property)
- MatchBy (see *MatchBy (property)* on page 170)
- Prepare() (see *Prepare() (method)* on page 208)
- ReportUnexpectedRows (see *ReportUnexpectedRows (property)* on page 220)
- rowIndex (see *rowIndex (property)* on page 224) (row property)
- tagName (see *tagName (property)* on page 279) (cell property)

wlTarget (property)

Property of Object

- wlHttp (see *wlHttp (object)* on page 316)

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 the *WebLOAD Scripting 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.

wlTimeParam() (parameterization)

Description

Return the current date and/or time according to the format specified in the parameter's properties

Syntax

```
<varName> = wlTimeParam(format, offset, usage)
```

Parameters

Parameter Name	Description
format	Formats are comprised from symbols, prefixed with the '%' sign and from textual characters. The following are the symbols used to create the date/time formats: • c – complete date time as number • H – Hours (24 hour clock) • I – Hours (12 hour clock) • M – minutes • S – seconds (S.000 – seconds with milliseconds) • p – AM or PM • d – day in month (number) • m – month (number) • y – year (2 digits) • Y – year (4 digits) • b – month name (3 letter) • B – month name (full) The following are the formats available to the user: • %c • %H:%M:%S • %I:%M:%S %p • %d-%b-%Y • %d/%m/%y • %m/%d/%y • %d/%m/%Y • %m/%d/%Y • %Y-%m-%d %H:%M:%S • %Y-%m-%d %H:%M:%S.000
offset	The offset in days and time. The offset can be used so the parameter will not consider the current date but another date in the future or in the past. A negative value indicates a date/time prior to current date/time.

Parameter Name	Description
usage	Defines when the parameter is updated, i.e., when a new value will be calculated. Possible values are: • wlParamUpdateRound. The parameters value will be calculated once for each round. Using the same parameter again in the same round will result in the same value. • wlParamUpdateOnce. The parameter's value will be calculated once per each Virtual Client (in its InitClient function). Every usage of the parameter by that Virtual Client will always result in the same value. • wlParamUpdateUse. The parameter's value will be calculated each time it is used.

Example

```
function InitClient()
{
NewParam1 = wlTimeParam("%Y-%m-%d %H:%M:%S", 1200,
wlParamUpdateRound);
}
/***** WLIDE - Message - ID:3 *****/
InfoMessage(NewParam1.getValue())

 // END WLIDE
```

wlVerification (object)

Description

The wlverification object stores the response validation properties set by the user through the WebLOAD Recorder, including HTML Web page title, text within a Web page, time taken to load a Web page, and size of a Web page (in bytes).

Note: Most global configuration property values and user-defined variables should be set through the WebLOAD Recorder. The property descriptions here are intended mainly to explain the lines of code seen in the JavaScript View of the WebLOAD Recorder desktop. Syntax details are also provided for the benefit of users who prefer to manually edit the JavaScript code of their scripts through the IntelliSense editor, described in *Using the IntelliSense JavaScript Editor* (on page 18).

Properties

The wlVerification object includes the following property classes:

- *Page Time*

- *Page Content Length*
- *Severity*
- *Function*
- *Error Message*

Methods

- *Title (function)*
- *ContentLength (function)*
- *MaxPageTime (function)*
- *Text (function)*

Syntax

Each individual property class includes the syntax specifications that apply to that class.

GUI mode

The `wlVerification` property and method descriptions explain how to explicitly set values for these response validations within your JavaScript script files.

The recommended way to set response validation values is through the WebLOAD Recorder, using the Response Validation dialog box accessed from the **Home** tab in the WebLOAD Recorder desktop ribbon. The dialog box provide a means of defining and setting response validation values with ease, simplicity, and clarity.

See also

- PageContentLength (see *PageContentLength (property)* on page 189)
- PageTime (see *PageTime (property)* on page 190)
- Severity (see *Severity (property)* on page 247)
- Function (see *Function (property)* on page 100)
- ErrorMessage (see *ErrorMessage (property)* on page 91)
- Title (see *Title (function)* on page 285)

wlVersion (property)

Property of Objects

- document (see *document (object)* on page 78)

Description

The HTTP protocol version, which WebLOAD retrieves from the HTTP header (read-only string, for example "1.1").

Example

```
currentVersionNumber = document.wlVersion
```

WLXmlDocument() (constructor)

Method of Object

- wlXmls (see *wlXmls (object)* on page 340)

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 website as described in the *WebLOAD Scripting 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

Parameter Name	Description
`[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

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- load() (see *load() (method)* on page 163)
- *load() and loadXML() Method Comparison* (on page 164)
- loadXML() (see *loadXML() (method)* on page 167)
- src (see *src (property)* on page 252)
- XMLDocument (see *XMLDocument (property)* on page 345)

wlXmls (object)

Property of Object

- document (see *document (object)* on page 78)

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 scripts 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 the *WebLOAD Scripting Guide*. For a complete list of the XML DOM properties and methods supported by WebLOAD, see *WebLOAD-supported XML DOM Interfaces* (on page 457).

Note: WebLOAD supports a new method for parsing and manipulating XML data. For more information see *XML Parser* Object on page 437.

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.wlXmls[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.wlXmls[0]
```

-Or-

```
MyBookstore = document.wlXmls.myXmlDoc
```

-Or-

```
MyBookstore = document.wlXmls[“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>
     JavaScript Reference Guide
       <author>Mark Twain</author>
       <title>Tom Sawyer</title>
       <price>$11.00</price>
     </book>
     JavaScript Reference Guide
       <author>Oscar Wilde</author>
       <title>The Giant And His Garden</title>
       <price>$8.00</price>
     </book>
  </bookstore>
</xml>
```

When accessing this website, your script 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.wlXmls[0]
InfoMessage(“ID = “ + XMLBookstoreDoc.id)
InfoMessage(“HTML text = “ + XMLBookstoreDoc.innerHTML)
```

Running this script 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 *WebLOAD-supported XML DOM Interfaces* (on page 457). These HTML properties and methods are accessed via the XMLDocument (see *XMLDocument (property)* on page 345) property. In addition, if the object is constructed from a Data Island, the id (see *id (property)* on page 146), InnerHTML (see *InnerHTML (property)* on page 154), and src (see *src (property)* on page 252) HTML properties are exposed. Each property is described in its own section.

- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)

- load() (see *load() (method)* on page 163)
- loadXML() (see *loadXML() (method)* on page 167)
- src (see *src (property)* on page 252)
- WLXmlDocument() (see *WLXmlDocument() (constructor)* on page 339)
- XMLDocument (see *XMLDocument (property)* on page 345)

See also

- *Collections* (on page 27)
- *load() and loadXML() Method Comparison* (on page 164)
- *XML Parser* Object on page 437

Write() (method)

Method of Object

- wlOutputFile (see *wlOutputFile (object)* on page 323)

Description

This method writes a string to the output file.

Syntax

```
Write(string)
```

Parameters

Parameter Name	Description
`string`	The text string you wish added to the output.

Example

```
MyFileObj = new wlOutputFile(filename)
…
MyFileObj.Write("Happy Birthday")
```

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page 28)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)

- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile() (see *wlOutputFile() (constructor)* on page 324)
- Writeln() (see *Writeln() (method)* on page 344)

Writeln() (method)

Method of Object

- wlOutputFile (see *wlOutputFile (object)* on page 323)

Description

This method writes a string followed by a newline character to the output file.

Syntax

```
Writeln(string)
```

Parameters

Parameter Name	Description
`string`	The text string you wish added to the output.

Example

```
MyFileObj = new wlOutputFile(filename)
…
MyFileObj.Writeln("Happy Birthday")
```

See also

- Close() (see *Close() (function)* on page 52)
- CopyFile() (see *CopyFile() (function)* on page 61)
- Delete() (see *Delete() (cookie method)* on page 75)
- *File Management Functions* (on page *28*)
- GetLine() (see *GetLine() (function)* on page 123)
- IncludeFile() (see *IncludeFile() (function)* on page 150)
- Open() (see *Open() (function)* on page 183)
- Reset() (see *Reset() (method)* on page 220)
- *Using the IntelliSense JavaScript Editor* (on page 18)
- wlOutputFile() (see *wlOutputFile() (constructor)* on page 324)
- Write() (see *Write() (method)* on page 343)

XMLDocument (property)

Method of Object

- wlXmls (see *wlXmls (object)* on page 340)

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 *WebLOAD-Supported XML DOM Interfaces* (on page 457).

Note: WebLOAD supports a new method for parsing and manipulating XML data. For more information see *XML Parser* Object on page 437.

Syntax

Use the following syntax:

`document.wlXmls[#].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.wlXmls[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.wlXmls[0].XMLDocument.documentElement.nodeName
```

See also

- *Collections* (on page 27)
- id (see *id (property)* on page 146)
- InnerHTML (see *InnerHTML (property)* on page 154)
- load() (see *load() (method)* on page 163)
- *load() and loadXML() Method Comparison* (on page 164)
- loadXML() (see *loadXML() (method)* on page 167)
- src (see *src (property)* on page 252)
- *XML Parser* Object on page 437

XMLParserObject (object)

Description

WebLOAD provides an embedded, third-party XML parser object to improve the multi-platform support for XML parsing within the WebLOAD environment. The XML parser object can be used instead of MSXML and Java XML parsing, resulting in lower memory consumption and increased performance during load testing.

The XML parser object can be used to reference any element in an XML document. For example, you can use the XML parser object to generate an Excel file containing the desired details of a specified element.

WebLOAD uses the Open Source Xerces XML parser (see http://xml.apache.org/xerces-c/).

The `parse()` method, not exposed by the original XML parser, is exposed by WebLOAD. This method is identical to the `parseURI()` method, except that it receives an XML string instead of a URI.

For more information on the XMLParserObject see *XML Parser* Object on page 437.

Syntax

The XML parser object is instanced as follows:

```
xmlObject = new XMLParserObject();
```

Example

For a detailed example of the implementation of the XML parser object, refer to *Example* on page 443.

Methods and Properties

- For a list of the XMLParserObject's methods, see *Methods* on page 438.
- For a list of the XMLParserObject's properties, see *Properties* on page 442.

Chapter 5

WebLOAD Internet Protocols Reference

This chapter provides detailed reference information on WebLOAD support for the following Internet protocols:

- FTP, through the wlFTP Object (on page 347) and wlFTPs Object (on page 359) (for secure SSL connections)
- HTML email, through the wlHtmMailer Object (on page 368)
- IMAP, through the wlIMAP Object (on page 374)
- NNTP, through the wlNNTP Object (on page 385)
- POP, through the wlPOP Object (on page 395) and wlPOPs Object (on page 402) (for secure SSL connections)
- SMTP, through the wlSMTP Object (on page 407) and wlSMTPs Object (on page 414) (for secure SSL connections)
- TCP, through the wlTCP Object (on page 419)
- Telnet, through the wlTelnet Object (on page 424)
- UDP, through the wlUDP Object (on page 430)

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 supported through the wlFTPs Object (on page 359).

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 `wlFTP` object connects to an FTP host.

```
ftp.PassWord = password
```

Caution: The password appears in plain text in the script. The password is visible to any user who has access to the script.

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 **0**. 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	Append(pattern)
Parameters	
pattern	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 Upload() method, Append() adds the data to the target file instead of overwriting it. If the target file does not exist, Append() creates it.

AppendFile()

Syntax	AppendFile(*filename*)
Parameters	
filename	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 `filename` 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 `filename` parameter. If the target file does not exist, AppendFile() creates it.

ChangeDir()

Syntax	ChangeDir(*name*)
Parameters	
name	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 `name` parameter.

ChFileMod()

Syntax	ChFileMod(*filename*, *attributes*)
Parameters	
filename	The name of the file you want to alter. This parameter may be a specific file name, or it may contain wildcards.
attributes	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 `attribute` parameter.

ChMod()

Syntax	ChMod(*pattern*, *attributes*)
Parameters	
pattern	The name of the files and directories you want to alter. This parameter may be a specific file name, or it may contain wildcards.
attributes	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 `attribute` 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	Delete(pattern)
Parameters	
pattern	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 DeleteFile() 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	Delete(filename)
Parameters	
filename	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	Dir(pattern)
Parameters	
pattern	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. 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 **Note:** If the host supports only basic information, only the fileName property of the array is defined.
Comments	Lists files and directories that match the pattern parameter in the current directory of the host. This method returns detailed information if the server supports it.

Download()

Syntax	Download(pattern)
Parameters	
pattern	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 property has been set, the data is saved to the specified file. If the Outfile 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	Download(*filename*)
Parameters	
filename	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 property has been set, the data is saved to the specified file. If the Outfile property has not been set, the file is saved with its current name.

GetCurrentPath()

Syntax	GetCurrentPath()
Return Value	A string containing the current path if successful, an exception if unsuccessful.
Comments	Returns the current path on the host.

GetStatusLine()

Syntax	GetStatusLine()
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	ListLocalFiles(filter)
Parameters	
`filter`	The files you want to list. The filter may be a patter or a specific file name.
Return Value	An array of matching objects with following properties if successful, an exception if unsuccessful. a[].fileName // A string containing name of the file a[].isDir // A Boolean, true if the entry represents a directory

Comments	Lists files matching the filter parameter in the current directory of the local computer.

Logoff()

Syntax	Logoff()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the FTP host.

Logon()

Syntax	Logon(*host*, [*port*])
Parameters	
`host`	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
`port`	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 UserName and PassWord properties before using this method.

MakeDir()

Syntax	MakeDir(*name*)
Parameters	
name	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	RemoveDir(*name*)
Parameters	
name	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	Rename(*from*, *to*)
Parameters	
from	The file that you want to rename.
to	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 `from` parameter to the name described in the `to` parameter.

SendCommand()

Syntax	SendCommand(*string*)
Parameters	
string	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	Upload(*pattern*)
Parameters	
pattern	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 `pattern` parameter to the host. The file is not renamed, and values specified by the `DataFile` and `Data` property are ignored. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files not transported.

UploadFile()

Syntax	UploadFile(*filename*)
Parameters	
filename	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 DataFile property. The destination file name is specified by the `filename` parameter. If the `DataFile` property is not specified, then the contents of the `Data` property are sent as a datastream to be saved under the name specified by the `filename` parameter.

UploadUnique()

Syntax	UploadUnique()
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 DataFile property. If the DataFile property is not specified, then the contents of the Data property are sent as a datastream.

WLFtp()

Syntax	new WLFtp()
Return Value	A new wlFTP object.
Comments	Creates a new wlFTP object, used to interact with the server.

Example	`function InitClient(){`
	`  myNewFtpObject = new WLFtp()`
	`}`

FTP Sample Code

```
// script 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 script. 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
```

wlFTPs Object

The `wlFTPs` object provides support for FTP (File Transfer Protocol) load and functional testing over secure connections (SSL).

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 `wlFTPs` object, you must include the `wlFtps.js` file in your `InitAgenda()` function.

wlFTPs 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 `wlFTP` object connects to an FTP host.

```
ftp.PassWord = password
```

Caution: The password appears in plain text in the script. The password is visible to any user who has access to the script.

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 **0**. 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
```

wlFTPs Methods

Append()

Syntax	Append(pattern)
Parameters	
pattern	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 Upload() method, Append() adds the data to the target file instead of overwriting it. If the target file does not exist, Append() creates it.

AppendFile()

Syntax	AppendFile(*filename*)
Parameters	
filename	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 `filename` 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 `filename` parameter. If the target file does not exist, AppendFile() creates it.

ChangeDir()

Syntax	ChangeDir(*name*)
Parameters	
name	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 `name` parameter.

ChFileMod()

Syntax	ChFileMod(*filename*, *attributes*)
Parameters	

filename	The name of the file you want to alter. This parameter may be a specific file name, or it may contain wildcards.
attributes	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 `attribute` parameter.

ChMod()

Syntax	ChMod(*pattern*, *attributes*)
Parameters	
pattern	The name of the files and directories you want to alter. This parameter may be a specific file name, or it may contain wildcards.
attributes	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 `attribute` 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	Delete(pattern)
Parameters	
pattern	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 DeleteFile() 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	Delete(filename)
Parameters	
filename	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	Dir(pattern)
Parameters	
pattern	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. 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 **Note:** If the host supports only basic information, only the fileName property of the array is defined.
Comments	Lists files and directories that match the pattern parameter in the current directory of the host. This method returns detailed information if the server supports it.

Download()

Syntax	Download(pattern)
Parameters	
pattern	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 property has been set, the data is saved to the specified file. If the Outfile 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	Download(*filename*)
Parameters	

filename	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 property has been set, the data is saved to the specified file. If the Outfile property has not been set, the file is saved with its current name.

GetCurrentPath()

Syntax	GetCurrentPath()
Return Value	A string containing the current path if successful, an exception if unsuccessful.
Comments	Returns the current path on the host.

GetStatusLine()

Syntax	GetStatusLine()
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	ListLocalFiles(filter)
Parameters	
`filter`	The files you want to list. The filter may be a patter or a specific file name.
Return Value	An array of matching objects with following properties if successful, an exception if unsuccessful. a[].fileName // A string containing name of the file a[].isDir // A Boolean, true if the entry represents a directory
Comments	Lists files matching the filter parameter in the current directory of the local computer.

Logoff()

Syntax	Logoff()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the FTP host.

Logon()

Syntax	Logon(*host*, [*port*])
Parameters	
`host`	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
`port`	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 UserName and PassWord properties before using this method.

MakeDir()

Syntax	MakeDir(*name*)
Parameters	
name	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	RemoveDir(*name*)
Parameters	
name	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	Rename(*from*, *to*)
Parameters	
from	The file that you want to rename.

to	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 `from` parameter to the name described in the `to` parameter.

SendCommand()

Syntax	SendCommand(*string*)
Parameters	
string	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	Upload(*pattern*)
Parameters	
pattern	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 `pattern` parameter to the host. The file is not renamed, and values specified by the `DataFile` and `Data` property are ignored. If an iteration of the loop fails, the loop is cancelled, potentially leaving some files not transported.

UploadFile()

Syntax	UploadFile(*filename*)
Parameters	
filename	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 DataFile property. The destination file name is specified by the `filename` parameter. If the `DataFile` property is not specified, then the contents of the `Data` property are sent as a datastream to be saved under the name specified by the `filename` parameter.

UploadUnique()

Syntax	UploadUnique()
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 DataFile property. If the DataFile property is not specified, then the contents of the Data property are sent as a datastream.

WLFtps()

Syntax	new WLFtps()
Return Value	A new wlFTPs object.
Comments	Creates a new wlFTPs object, used to interact with the server.

Example	`function InitClient(){`
	`  myNewFtpsObject = new WLFtps()`
	`}`

wlHtmMailer Object

The `wlHtmMailer` object provides support for HTM Mail load and functional testing within WebLOAD. Support for standard HTM Mail operation is included. HTM Mail 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 `wlHtmMailer` object. If you do not, the object may cause your script to freeze. A sample catch appears in the `wlHtmMailer` code sample at the end of this section.

To access the `wlHtmMailer` object, you must include the `wlHtmMailer.js` file in your `InitAgenda()` function.

wlHtmMailer Properties

AttachmentsArr

The `AttachmentsArr` property lets you specify one or more attachments for an email. The `filename` variable should contain the name of the local file or datastream that you want to attach to the posting. For example:

```
wlHtmMailer.Attachments[0] = filename
```

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:

```
wlHtmMailer.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:

```
wlHtmMailer.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:

```
wlHtmlMailer.From = replyname
```

Host

The `Host` property lets you specify a host for use in sending HTML email messages.

HtmlFilePath

The `HtmlFilePath` property specifies the full path directory location of the files associated with the email message.

HtmlText

The `HtmlText` property contains the HTML-formatted version of the email message, for example, potentially including embedded images. The corresponding plain text version of the email message is provided in the `Message` property.

Message

The `Message` property contains the plain text version of the email message. If there is a corresponding HTML-formatted version, for example, including embedded images, this version is provided in the `HtmlText` property.

MessageDate

The `MessageDate` property contains the date of the email message.

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:

```
wlHtmMailer.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 = wlHtmMailer.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:

```
wlHtmMailer.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:

```
wlHtmMailer.To = recipientaddress; recipientaddress
```

wlHtmMailer Methods

AddAttachment()

Syntax	AddAttachment(*string*, *type*, [*encoding*])
Parameters	
string	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.
type	The type of attachment you are sending. Valid values are: `File` (default) `Data`

[encoding]	The type of encoding to apply to the file. Valid values are:
	`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	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
[port]	The port to which you are connecting. If you do not specify a port, the default session port is used.
Return Value	Null if successful, an exception if unsuccessful.

Comments	Starts a new session with the host.

DeleteAttachment()

Syntax	DeleteAttachment(*string*)
Parameters	
string	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()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the host.

DisplayMetrics()

Syntax	DisplayMetrics()
Return Value	A string with the current metrics values.
Comments	Displays all the information gathered from the last command or data transfer.

GetLocalHost()

Syntax	GetLocalHost()
Return Value	Identification information for the currently active local host.
Comments	Returns identification information for the current local host.

GetStatusLine()

Syntax	GetStatusLine()
Return Value	A string containing the latest response string if successful, an exception if unsuccessful.
Comments	Returns the latest response string from the host.

Send()

Syntax	Send()
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(*string*)
Parameters	
string	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.

SetLocalHost()

Syntax	SetLocalHost(*hostname*)
Parameters	
hostname	Identification information for the new local host.
Return Value	Assigns a new value for the local host.
Comments	Defines the local host from which the emails are being sent.

Verify()

Syntax	Verify()
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.

WLHtmMailer()

Syntax	new WLHtmMailer()
Return Value	A new wlHtmMailer object.
Comments	Creates a new wlHtmMailer object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewHtmMailerObject = new WLHtmMailer();`
	`}`

wlIMAP Object

The `wlIMAP` 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 `wlIMAP` object, you must include the `wlImap.js` file in your `InitAgenda()` function.

wlIMAP 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 wlIMAP 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 wlIMAP object connects to an IMAP host. For example:

```
imap.UserName = username
```

wlIMAP Methods

Connect()

Syntax	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
port	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()
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 script. To remove a mailbox, use the `DeleteMailbox()` method.

Delete()

Syntax	Delete([*MessageSet*])
Parameters	
MessageSet	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()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Deletes the mailbox specified in the `Mailbox` property.

Disconnect()

Syntax	Disconnect()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the IMAP host.

GetMessageCount()

Syntax	GetMessageCount()
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	GetStatusLine()
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	ListMailboxes(pattern)
Parameters	
pattern	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 `pattern` parameter.

RecentMessageCount()

Syntax	RecentMessageCount()
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	RenameMailbox(*string*)
Parameters	
string	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	Retrieve([*MessageSet*])
Parameters	
MessageSet	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	Search(*string*)
Parameters	
string	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. **UNFLAGGED** – Messages that do not have the \\Flagged flag set. **UNKEYWORD** – Messages that do not have the specified keyword set. **UNSEEN** – Messages that do not have the \\Seen 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	SendCommand(*string*)
Parameters	
string	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	SubscribeMailbox()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Subscribes to the mailbox specified in the `Mailbox` property.

UnsubscribeMailbox()

Syntax	UnsubscribeMailbox()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Unsubscribes from the mailbox specified in the `Mailbox` property.

WLImap()

Syntax	new WLImap()
Return Value	A new wlIMAP object.
Comments	Creates a new wlIMAP object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewImapObject = new WLImap()`
	`  myNewImapObject.Connect("HostName")`
	`}`

IMAP Sample Code

```
// script 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 script
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")
```

wlNNTP Object

The `wlNNTP` (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 `wlNNTP` object. If you do not, the object may cause your script to freeze. A sample catch appears in the NNTP code sample at the end of this section.

To access the `wlNNTP` object, you must include the `wlNntp.js` file in your `InitAgenda()` function.

wlNNTP 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 an 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 `wlNNTP` object connects to an NNTP host. For example:

```
nntp.PassWord = password
```

Caution: The password appears in plain text in the script. The password is visible to any user who has access to the script.

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 wlNNTP object connects to an NNTP host. For example:

```
nntp.UserName = username
```

wlNNTP Methods

AddAttachment()

Syntax	AddAttachment(*string*, *type*, [*encoding*])
Parameters	
string	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.
type	The type of attachment you are sending. Valid values are: `File` (default) `Data`

[encoding]	The type of encoding to apply to the file. Valid values are:
	`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	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
[port]	The port to which you are connecting. If you do not specify a port, the default session port is used.
Return Value	Null if successful, an exception if unsuccessful.
Comments	Starts a new session with the host.

DeleteAttachment()

Syntax	DeleteAttachment(*string*)
Parameters	
string	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()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the host.

GetArticle()

Syntax	GetArticle(*messageNumber*)
Parameters	
message Number	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 `messageNumber` parameter for the group specified in the `Group` property. If the `Outfile` property is specified, the returned article is stored in the output file as well as in the document property.

GetArticleCount()

Syntax	GetArticleCount()
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 `Group` property.

GetStatusLine()

Syntax	GetStatusLine()
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	GroupOverview([*range*])
Parameters	
[range]	The range for articles you want to view. The format for `range` 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 `articleDate`, `articleLines`, `articleNumber`, `from`, `messageID`, `otherHeaders`, `references`, and `subject`. The method returns an exception if unsuccessful.
Comments	Returns an overview for the articles in range for the group specified in the `Group` property.

ListGroups()

Syntax	ListGroups([*startDate*])
Parameters	

[startDate]	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 `startDate` is YYMMDD HHMMSS.
Return Value	An array of objects if successful. Each object contains the following properties, `Canpost`, `lastArticle`, `firstArticle`, and `group`. The method returns an exception if unsuccessful.
Comments	Lists the newsgroups available on the host.

PostArticle()

Syntax	PostArticle()
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:
	Header Properties
	`From`
	`Subject`
	`Organization`
	`To`
	`ReplyTo`
	`References`
	`MaxHeadersLength`
	Body Properties/Methods
	`ArticleText`()
	`AddAttachment`()
	`DeleteAttachment`()

SendCommand()

Syntax	SendCommand(*string*)
Parameters	
string	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	new WLNntp()
Return Value	A new wlNNTP object.
Comments	Creates a new wlNNTP object, used to interact with the server.
Example	`myNewNntpObject = new WLNntp()`

NNTP Sample Code

```
// script Initialization
function InitAgenda() {
  IncludeFile("wlNntp.js",WLExecuteScript)
}
//=====================================================

//Body Of script
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:\\downloded.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")
```

wlPOP 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 supported through the wlPOPs Object (on page 402).

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.

wlPOP 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 wlPOP object connects to a POP host. For example:

```
pop.PassWord = password
```

Caution: The password appears in plain text in the script. The password is visible to any user who has access to the script.

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 a 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	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
[port]	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	Delete([*MessageID*])
Parameters	
messageID	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	Disconnect()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the POP server.

GetCurrentMessageID()

Syntax	GetCurrentMessageID()
Return Value	The ID of the current message if successful, an exception if unsuccessful.
Comments	Returns the ID of the current message.

GetMailboxSize()

Syntax	GetMailboxSize()
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	GetMessageCount()
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	GetStatusLine()
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	Reset()
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	Retrieve([*MessageID*])
Parameters	
MessageID	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(*string*)
Parameters	
string	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()
Return Value	A new wlPOP object.
Comments	Creates a new wlPOP object, used to interact with the server.
Example	`var myNewPopObject = new WLPop();`

POP Sample Code

```
// script Initialization
function InitAgenda() {
  IncludeFile("wlPop.js",WLExecuteScript)
}
/*function InitClient() {

}*/
/*function TerminateClient() {
  delete pop;
}*/
//=====================================================

//Body Of script.
//InfoMessage("Speed: "+wlGlobals.ConnectionSpeed)
wlGlobals.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.wlSource);
}*/
catch (e)
{
  InfoMessage ("Error" + e)
}
pop.Disconnect();
//============================================================
```

wlPOPs Object

The `wlPOPs` object provides support for POP3 (Post Office Protocol) load and functional testing over secure connections (SSL).

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 `wlPOPs` object, you must include the `wlPops.js` file in your `InitAgenda()` function.

wlPOPs 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 wlPOP object connects to a POP host. For example:

```
pop.PassWord = password
```

Caution: The password appears in plain text in the script. The password is visible to any user who has access to the script.

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 a 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
```

wlPOPs Methods

Connect()

Syntax	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may describe the host using its DNS number, or by giving its name.
[port]	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	Delete([*MessageID*])
Parameters	
messageID	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	Disconnect()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the POP server.

GetCurrentMessageID()

Syntax	GetCurrentMessageID()

Return Value	The ID of the current message if successful, an exception if unsuccessful.
Comments	Returns the ID of the current message.

GetMailboxSize()

Syntax	GetMailboxSize()
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	GetMessageCount()
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	GetStatusLine()
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	Reset()
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	Retrieve([*MessageID*])
Parameters	
MessageID	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(*string*)
Parameters	
string	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.

WLPops()

Syntax	new WLPops()
Return Value	A new wlPOPs object.
Comments	Creates a new wlPOPs object, used to interact with the server.
Example	`var myNewPopObject = new WLPops();`

wlSMTP Object

The `wlSMTP` object provides support for Simple Mail Transfer Protocol (SMTP) load and functional testing within WebLOAD. Support for standard SMTP operation is included. SMTP over secure connections (SSL) is supported through the *wlSMTPs Object* (on page 414).

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 `wlSMTP` object, you must include the `wlSmtp.js` file in your `InitAgenda()` function.

wlSMTP 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:

- `WLSmtp.ENC_7BIT` – 7bit encoding
- `WLSmtp.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 STMP server
- **ESMTP** – An extended SMTP server

For example:

```
smtp.Type = servertype
```

wlSMTP Methods

AddAttachment()

Syntax	AddAttachment(*string*, *type*, [*encoding*])
Parameters	
String	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.

Type	The type of attachment you are sending. The default value is `File`. Valid values are: • `File` • `Data`
encoding	The type of encoding to apply to the file. The default value is `7Bit`. Valid values are: • `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	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
port	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(*ID*)
Parameters	
ID	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()
Return Value	Null if successful, an exception if unsuccessful.

Comments	Terminates a connection to the SMTP host.

Send()

Syntax	Send()
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(*string*)
Parameters	
string	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. SendCommand automatically appends "\r\n" at the end of the string. You can add additional instances of "\r\n" within the string, however do not add "\r\n" at the end of the string. For example, SendCommand("Line1\r\n Line2\r\n Line3")

Verify()

Syntax	Verify()
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()
Return Value	A new wlSMTP object.
Comments	Creates a new wlSMTP object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewSmtpObject = new WLSmtp()`
	`}`

SMTP Sample Code

```
// script 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 script
//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:\\downloded.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
catch (e)
{
  InfoMessage ("Error" + e)
}
//=======================================================
InfoMessage("done")       // End of SMTP sample script
```

wlSMTPs Object

The `wlSMTP` object provides support for SMTP (Mail Transfer Protocol) load and functional testing over secure connections (SSL).

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 `wlSMTPs` object, you must include the `wlSMTPs.js` file in your `InitAgenda()` function.

wlSMTPs 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:

- `WLSmtp.ENC_7BIT` – 7bit encoding
- `WLSmtp.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 STMP server
- **ESMTP** – An extended SMTP server

For example:

```
smtp.Type = servertype
```

wlSMTPs Methods

AddAttachment()

Syntax	AddAttachment(*string*, *type*, [*encoding*])
Parameters	
String	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.
Type	The type of attachment you are sending. The default value is `File`. Valid values are: • `File` • `Data`
encoding	The type of encoding to apply to the file. The default value is `7Bit`. Valid values are: • `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	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.

port	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(*ID*)
Parameters	
ID	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()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the SMTP host.

Send()

Syntax	Send()
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(*string*)
Parameters	
string	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()

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.

WLSmtps()

Syntax	new WLSmtps()
Return Value	A new wlSMTPs object.
Comments	Creates a new wlSMTPs object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewSmtpObject = new WLSmtps()`
	`}`

wlTCP Object

The `wlTCP` 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 `wlTCP` object, you must include the `wlTcp.js` file in your `InitAgenda()` function.

wlTCP 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 script 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 script 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.

For example:

```
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 script breaks its connection with the host. If you do not specify the timeout property, the script 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 scripts that use the `wlTCP` object. If you do not, and the script fails to return a prompt, your session may freeze.

wlTCP Methods

Connect()

Syntax	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
port	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	Disconnect()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Terminates a connection to the TCP host.

Erase()

Syntax	Erase()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Clears the contents of the document object.

Receive()

Syntax	Receive()
Return Value	Null if successful, an exception if unsuccessful.

Comments	Returns all responses from the host since the last time the `Send()` method was used. A `Receive()` method returns to the script when the `NextPrompt`, `NextSize`, or `Timeout` properties are met. If more than one of these properties is specified, the method returns to the script when the first one is met. Subsequent uses of `Receive()` 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	Send(data_to_send)
Parameters	
data_to_send	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	new WLTcp()
Return Value	A new wlTCP object.
Comments	Creates a new wlTCP object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewTcpObject = new WLTcp();`
	`}`

TCP Sample Code

```
// script Initialization
function InitAgenda() {
  IncludeFile("wlTcp.js",WLExecuteScript)
}
function InitClient() {
  tcp=new WLTcp();
}
function TerminateClient()
{
  delete tcp;
}
//=================================================
```

```
//Body Of script.
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");
```

wlTelnet Object

The `wlTelnet` 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 `wlTelnet` object, you must include the `wlTelnet.js` file in your `InitAgenda()` function.

wlTelnet 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 script breaks its connection with the host. If you do not specify the timeout property, the script 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 scripts that use the `wlTelnet` object. If you do not, and the script fails to return a prompt, your session may freeze.

wlTelnet Methods

Connect()

Syntax	Connect(*host*, [*port*])
Parameters	
host	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
port	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	Disconnect()
Return Value	Null if successful, an exception if unsuccessful
Comments	Terminates a connection to the Telnet host.

Erase()

Syntax	Erase()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Clears the contents of the document object.

Receive()

Syntax	Receive()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Returns all responses from the host since the last time the `Send()` method was used. A `Receive()` method returns to the script when the `NextPrompt`, `NextSize`, or `Timeout` properties are met. If more than one of these properties is specified, the method returns to the script when the first one is met. Subsequent uses of `Receive()` 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	Send(data_to_send)
Parameters	
data_to_send	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	new WLTelnet()
Return Value	A new wlTelnet object.
Comments	Creates a new wlTelnet object, used to interact with the server.

Example	function **InitClient()** {
	  myNewTelnetObject = new WLTelnet()
	}

Telnet Sample Code

```
// script 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 script
// Set timeout and prompt
// IMPORTANT: Set a timeout when setting a prompt. Otherwise,
// If the prompt is unexpected or incorrect the script 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 script 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
```

wlUDP Object

The `wlUDP` 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 `wlUDP` object, you must include the `wlUdp.js` file in your `InitAgenda()` function.

wlUDP 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 script.

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 script breaks its connection with the host. If you do not specify the timeout property, the script 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 scripts that use the `wlUDP` object. If you do not, and the script fails to return a prompt, your session may freeze.

wlUDP Methods

Bind()

Syntax	Bind()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Creates a UDP port and sets the `OutBufferSize`, `InBufferSize`, `MaxDatagramSize`, `LocalHost`, and `LocalPort` properties. The value of these properties is fixed when the `Bind()` method is used. To change the value of any of these properties, you must use the `UnBind()` method, change the value of the property and using the `Bind()` method again.

Broadcast()

Syntax	Broadcast(*port, data_to_send*)
Parameters	
Port	The port to which you are connecting.
data_to_send	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	Erase()
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	Receive()
Return Value	Null if successful, an exception if unsuccessful.
Comments	Returns all responses from the host since the last time the `Send()` method was used. The `Receive()` method returns to the script when the `RequestedPackets` or `Timeout` property is met. Subsequent uses of `Receive()` find the next instance of the limiting property, returning additional information from the buffer.

Send()

Syntax	Send(*host, port, data_to_send*)
Parameters	
Host	The host to which you are connecting. You may express the host using either the DNS number or the full name of the host.
port	The port to which you are connecting.
data_to_send	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	UnBind()
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 `Bind()` again.

WLUdp()

Syntax	new WLUdp()
Return Value	A new wlUDP object.
Comments	Creates a new wlUDP object, used to interact with the server.

Example	`function InitClient() {`
	`  myNewUDPObject = new WLUdp()`
	`}`

UDP Sample Code

```
// script 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 script.
//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
```

Chapter 6

XML Parser Object

WebLOAD provides an embedded, third-party XML parser object to improve the multi-platform support for XML parsing within the WebLOAD environment. The XML parser object can be used instead of MSXML and Java XML parsing, resulting in lower memory consumption and increased performance during load testing.

The XML parser object can be used to reference any element in an XML document. For example, you can use the XML parser object to generate an Excel file containing the desired details of a specified element.

WebLOAD uses the Open Source Xerces XML parser (see http://xml.apache.org/xerces-c/).

The XML parser object is instanced as follows:

```
xmlObject = new XMLParserObject();
```

The `parse()` method, not exposed by the original XML parser, is exposed by WebLOAD. This method is identical to the `parseURI()` method, except that it receives an XML string instead of a URI.

The following sections provide lists of exposed methods and properties as well as a detailed example of the implementation of the XML parser object.

Note: For additional information, refer to:
http://xml.apache.org/xerces-c/ApacheDOMC++BindingL2.html

Methods

The following table lists the XML parser object methods exposed by WebLOAD.

Table 6. XML Parser Object Methods

Object	Method Name
xmlparser	• parseURI • parse • resetDocumentPool • release • setFeature • getFeature • canSetFeature • load • loadXML
xmlAttr	• getName • getValue • setValue • getOwnerElement • getSpecified
xmlCharacterData	• getData • getLength • appendData • setData • substringData • deleteData • insertData • replaceData

Object	Method Name
xmlDocument	• createElement • getElementById • getDocumentElement • getElementsByTagName • createTextNode • createDocumentFragment • getDoctype • createComment • createCDATAsection • createAttribute • createEntityReference • createProcessingInstruction • createElementNS • createAttributeNS • getElementsByTagNameNS • importNode
xmlDocumentType	• getName • getPublicId • getSystemId • getInternalSubset • getEntities • getNotations

Object	Method Name
xmlElement	• getElementsByTagName • getElementsByTagNameNS • getAttribute • getAttributeNS • getAttributeNode • getAttributeNodeNS • setAttributeNode • setAttributeNodeNS • getTagName • hasAttribute • hasAttributeNS • removeAttribute • removeAttributeNS • setAttribute • setAttributeNS • removeAttributeNode
xmlEntity	• getPublicId • getSystemId • getNotationName
xmlnamednodemap	• getLength • getNamedItem • removeNamedItem • getNamedItemNS • removeNamedItemNS • setNamedItem • setNamedItemNS • item

Object	Method Name
xmlnode	• getNodeName • getNodeValue • getNodeType • getParentNode • getFirstChild • getLastChild • getPreviousSibling • getNextSibling • getChildNodes • getAttributes • getOwnerDocument • getNamespaceURI • getPrefix • getLocalName • hasChildNodes • hasAttributes • normalize • release • removeChild • appendChild • insertBefore • setNodeValue • setPrefix • isSupported
xmlnodelist	• Item • getLength
xmlNotation	• getPublicId • getSystemId • xmlProcessingInstruction • getTarget • getData • setData

Properties

The following table lists the XML parser object properties exposed by WebLOAD.

Table 7. XML Parser Object Properties

Object	Property Name
xmlAttr	• name • value
xmlCharacterData	• length • data
xmlDocument	• documentElement
xmlElement	• tagName
xmlnode	• nodeName • attributes • childNodes • firstChild • lastChild • namespaceURI • nextSibling • nodeType • nodeValue • ownerDocument • parentNode • prefix • previousSibling • nodeTypeString • xml
xmlnodelist	• length
xmlProcessingInstruction	• target • Data

Example

The following is an example of the use of the XML parser object:

```
{
   //Create the XML parser object (xerces-c parser)
   xmlObject = new XMLParserObject();

   //Parse the xml file from the specified path
   xmlDoc  = xmlObject.parseURI("C:\\xml_file.xml");

   //Retrieve the first node with the “NODE5” tag
   domNode = xmlDoc.getElementsByTagName("NODE5").item(0);

   //Retrieve the node's type
   nodeType = domNode.getNodeType();

   //Retrieve the node's parent
   nodeParent = domNode.getParentNode().getNodeName();

   //Retrieve the number of child nodes
   numOfChilds = domNode.getChildNodes().getLength();

   //Create a new element
   newNode1 = xmlDoc.createElement("NEW_NODE1");

   //Insert the new element into DOM
   domNode1.insertBefore(newNode1, domNode);
```

Chapter 7

WebSocket Object

WebLOAD supports WebSocket, a protocol that provides full-duplex communication channels over a single TCP connection.

Unlike HTTP which is a request-response protocol, WebSocket creates connections for sending or receiving messages that are not dependent on one another. In this way, WebSocket provides full-duplex communication. WebSocket also enables streams of messages on top of TCP.

WebLOAD's WebSocket object enables creating and managing a WebSocket connection to a server, as well as sending and receiving data on the connection.

Note that you can create multiple WebSocket objects.

Constructor

Description

Creates a new WebSocket for the given URL, and returns a JavaScript object reference.

Syntax

```
<websocket object name> = new WebSocket (<URL>);
```

Parameters

Parameter Name	Description
URL	The URL to which to connect.

Example

```
ws1 = new WebSocket("ws://echo.websocket.org");
```

Methods

connect() (method)

Description

Creates a WebSocket connection to the given URL address. When connected, an `onopen()` event is fired, as described in *onopen (evt)* (on page 447).

Syntax

<websocket object name>.**connect()**

Example

```
ws1.connect()
```

close() (method)

Description

Closes the WebSocket connection.

Syntax

<websocket object name>.**close()**

Example

```
ws1.close()
```

send() (method)

Description

Sends data to a WebSocket connection.

Syntax

<websocket object name>.**send(data[,encoded])**

Parameters

Parameter Name	Description
`data`	The data to be sent, enclosed in quote marks.
`[encoded]`	An optional Boolean value (true or false). • True indicates that the data contains an ASCII encoded string in the format %xx, where xx is the hexadecimal ASCII code. • False indicates the data does not contain an ASCII encoded string. This is the default value.

Examples

```
ws1.send("hi")
ws1.send("next line %0A here", true)
```

Events

A WebSocket emits events. An Event handler should be registered in order to react to events.

onmessage (evt)

An event that occurs when a new message is received.

- `evt.getData()` – Gets the data. This can be a string or binary data.
- `evt.isBinary()` – Indicates whether the data is binary or not.
- `evt.getEncodedData()` – Gets the data in encoded format. This is useful for binary messages.

Example

```
ws1.onmessage = function(evt) {
    InfoMessage("got message " + evt.getData() )
    if (evt.isBinary() ) {
    InfoMessage("Message is binary");
    }
}
```

onerror (evt)

An event that occurs when an error message is received. The default behavior is to show a warning message with the error details.

- `Evt.getData()` – gets the underlying exception details.

onopen (evt)

An event that occurs when the socket is opened (connected).

Example

```
ws1.onopen = function(evt) {
    DebugMessage("WebSocket is opened, say hello");
    ws1.send("hello");
}
```

WebSocket Sample Code

```
// Create a WebSocket object
ws1 = new WebSocket( "ws://echo.websocket.org" );
// Define an event handler, to handle events (incoming messages)
when they occur
 ws1.onmessage = function(evt) {
    // Display in the Log the text that was sent in the message body
    DebugMessage("Server said:" + evt.getData());
}
// Create a websocket connection
ws1.connect();
//Note that events are handled while in Sleep
Sleep(1000);
// Send a message with the text “hi”
ws1.send("hi");
Sleep(1000);
// Close the websocket connection
ws1.close();
Sleep(1000);
```

Appendix A

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:

SSL handshake combinations

Table 8: 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 Ciphers – Complete List

WebLOAD's SSL support is based on the OpenSSL open source project (http://www.openssl.org/). Table 9 contains a complete list of ciphers supported by WebLOAD using OpenSSL. Abbreviations used in this list are explained in Table 10.

For information on how WebLOAD provides full SSL/TLS 1.0/TLS 1.2 protocol support through the Cipher Command Suite, see *SSL Cipher Command Suite* on page 33.

The following table lists all ciphers supported by WebLOAD.

Table 9: SSL Ciphers Supported by WebLOAD

Name	Mode	Key Ex.	Auth.	Encryption method (key length)	Message digest algorithm	Export
TLS1_CK_SRP_SHA_WITH_AES_256_CBC_SHA	TLS1.2	SRP	SHA	AES(256)-CBC	SHA	
TLS1_CK_SRP_SHA_RSA_WITH_AES_256_CBC_SHA	TLS1.2	SRP	SHA	AES(256)-CBC	SHA	
TLS1_CK_RSA_WITH_AES_128_SHA256	TLS1.2	RSA	RSA	AES(128)	SHA256	
TLS1_CK_RSA_WITH_AES_256_SHA256	TLS1.2	RSA	RSA	AES(256)	SHA256	
TLS1_CK_DH_DSS_WITH_AES_128_SHA256	TLS1.2	DH	DDS	AES(128)	SHA256	
TLS1_CK_DH_RSA_WITH_AES_128_SHA256	TLS1.2	DH	RSA	AES(128)	SHA256	
TLS1_CK_DHE_DSS_WITH_AES_128_SHA256	TLS1.2	DHE	DDS	AES(128)	SHA256	
TLS1_CK_DHE_RSA_WITH_AES_128_SHA256	TLS1.2	DHE	RSA	AES(128)	SHA256	
TLS1_CK_DH_DSS_WITH_AES_256_SHA256	TLS1.2	DH	DDS	AES(256)	SHA256	
TLS1_CK_DH_RSA_WITH_AES_256_SHA256	TLS1.2	DH	RSA	AES(256)	SHA256	
TLS1_CK_DHE_DSS_WITH_AES_256_SHA256	TLS1.2	DHE	DDS	AES(256)	SHA256	
TLS1_CK_DHE_RSA_WITH_AES_256_SHA256	TLS1.2	DHE	RSA	AES(256)	SHA256	
TLS1_CK_ADH_WITH_AES_128_SHA256	TLS1.2	ADH	None	AES(128)	SHA256	
TLS1_CK_ADH_WITH_AES_256_SHA256	TLS1.2	ADH	None	AES(256)	SHA256	
TLS1_CK_RSA_WITH_AES_128_GCM_SHA256	TLS1.2	RSA	RSA	AES(128)-GCM	SHA256	
TLS1_CK_RSA_WITH_AES_256_GCM_SHA384	TLS1.2	RSA	RSA	AES(256) -GCM	SHA384	
TLS1_CK_DHE_RSA_WITH_AES_128_GCM_SHA256	TLS1.2	DH	RSA	AES(128) -GCM	SHA256	
TLS1_CK_DHE_RSA_WITH_AES_256_GCM_SHA384	TLS1.2	DH	RSA	AES(256) -GCM	SHA384	
TLS1_CK_DH_RSA_WITH_AES_128_GCM_SHA256	TLS1.2	DH	RSA	AES(128) -GCM	SHA256	
TLS1_CK_DH_RSA_WITH_AES_256_GCM_SHA384	TLS1.2	DH	RSA	AES(256-GCM	SHA384	
TLS1_CK_DHE_DSS_WITH_AES_128_GCM_SHA256	TLS1.2	DHE	DDS	AES(128) -GCM	SHA256	
TLS1_CK_DHE_DSS_WITH_AES_256_GCM_SHA384	TLS1.2	DHE	DDS	AES(256) -GCM	SHA384	
TLS1_CK_DH_DSS_WITH_AES_128_GCM_SHA256	TLS1.2	DH	DDS	AES(128) -GCM	SHA256	

Name	Mode	Key Ex.	Auth.	Encryption method (key length)	Message digest algorithm	Export
TLS1_CK_DH_DSS_WITH_AES_256_GCM_SHA384	TLS1.2	DH	DDS	AES(256) -GCM	SHA384	
TLS1_CK_ADH_WITH_AES_128_GCM_SHA256	TLS1.2	ADH	None	AES(128) -GCM	SHA256	
TLS1_CK_ADH_WITH_AES_256_GCM_SHA384	TLS1.2	ADH	None	AES(256) -GCM	SHA384	
AECDH-DES-CBC3-SHA	TLS1	ECDH	None	3DES(168)	SHA1	
ECDHE-RSA-DES-CBC3-SHA	TLS1	ECDH	RSA	3DES(168)	SHA1	
ECDH-RSA-DES-CBC3-SHA	TLS1	ECDH	RSA	3DES(168)	SHA1	
ECDHE-ECDSA-DES-CBC3-SHA	TLS1	ECDH	ECDSA	3DES(168)	SHA1	
ECDH-ECDSA-DES-CBC3-SHA	TLS1	ECDH	ECDSA	3DES(168)	SHA1	
ADH-DES-CBC3-SHA	SSLv3	DH	None	3DES(168)	SHA1	
EDH-RSA-DES-CBC3-SHA	SSLv3	DH	RSA	3DES(168)	SHA1	
EDH-DSS-DES-CBC3-SHA	SSLv3	DH	DSS	3DES(168)	SHA1	
DH-RSA-DES-CBC3-SHA	SSLv3	DH/RSA	DH	3DES(168)	SHA1	
DH-DSS-DES-CBC3-SHA	SSLv3	DH/DSS	DH	3DES(168)	SHA1	
DES-CBC3-SHA–SSL3	SSLv3	RSA	RSA	3DES(168)	SHA1	
DES-CBC3-MD5	SSLv2	RSA	RSA	3DES(168)	MD5	
DES-CBC3-SHA–SSL2	SSLv2	RSA	RSA	3DES(192)	SHA1	
AECDH-AES128-SHA	TLS1	ECDH	None	AES(128)	SHA1	
ECDHE-RSA-AES128-SHA	TLS1	ECDH	RSA	AES(128)	SHA1	
ECDH-RSA-AES128-SHA	TLS1	ECDH	RSA	AES(128)	SHA1	
ECDHE-ECDSA-AES128-SHA	TLS1	ECDH	ECDSA	AES(128)	SHA1	
ECDH-ECDSA-AES128-SHA	TLS1	ECDH	ECDSA	AES(128)	SHA1	
ADH-AES128-SHA	TLS1	DH	None	AES(128)	SHA1	
DHE-RSA-AES128-SHA	TLS1	DH	RSA	AES(128)	SHA1	
DHE-DSS-AES128-SHA	TLS1	DH	DSS	AES(128)	SHA1	
DH-RSA-AES128-SHA	TLS1	DH/RSA	DH	AES(128)	SHA1	
DH-DSS-AES128-SHA	TLS1	DH/DSS	DH	AES(128)	SHA1	
AES128-SHA	TLS1	RSA	RSA	AES(128)	SHA1	
AECDH-AES256-SHA	TLS1	ECDH	None	AES(256)	SHA1	
ECDHE-RSA-AES256-SHA	TLS1	ECDH	RSA	AES(256)	SHA1	
ECDH-RSA-AES256-SHA	TLS1	ECDH	RSA	AES(256)	SHA1	
ECDHE-ECDSA-AES256-SHA	TLS1	ECDH	ECDSA	AES(256)	SHA1	
ECDH-ECDSA-AES256-SHA	TLS1	ECDH	ECDSA	AES(256)	SHA1	
ADH-AES256-SHA	TLS1	DH	None	AES(256)	SHA1	

Name	Mode	Key Ex.	Auth.	Encryption method (key length)	Message digest algorithm	Export
DHE-RSA-AES256-SHA	TLS1	DH	RSA	AES(256)	SHA1	
DHE-DSS-AES256-SHA	TLS1	DH	DSS	AES(256)	SHA1	
DH-RSA-AES256-SHA	TLS1	DH/RSA	DH	AES(256)	SHA1	
DH-DSS-AES256-SHA	TLS1	DH/DSS	DH	AES(256)	SHA1	
AES256-SHA	TLS1	RSA	RSA	AES(256)	SHA1	
EXP-ADH-DES-CBC-SHA	SSLv3	DH(512)	None	DES(40)	SHA1	☑
EXP-EDH-RSA-DES-CBC-SHA	SSLv3	DH(512)	RSA	DES(40)	SHA1	☑
EXP-EDH-DSS-DES-CBC-SHA	SSLv3	DH(512)	DSS	DES(40)	SHA1	☑
EXP-DH-RSA-DES-CBC-SHA	SSLv3	DH/RSA	DH	DES(40)	SHA1	☑
EXP-DH-DSS-DES-CBC-SHA	SSLv3	DH/DSS	DH	DES(40)	SHA1	☑
EXP-DES-CBC-SHA	SSLv3	RSA(512)	RSA	DES(40)	SHA1	☑
EXP1024-DHE-DSS-DES-CBC-SHA	TLS1	DH(1024)	DSS	DES(56)	SHA1	☑
EXP1024-DES-CBC-SHA	TLS1	RSA(1024)	RSA	DES(56)	SHA1	☑
ADH-DES-CBC-SHA	SSLv3	DH	None	DES(56)	SHA1	
EDH-RSA-DES-CBC-SHA	SSLv3	DH	RSA	DES(56)	SHA1	
EDH-DSS-DES-CBC-SHA	SSLv3	DH	DSS	DES(56)	SHA1	
DH-RSA-DES-CBC-SHA	SSLv3	DH/RSA	DH	DES(56)	SHA1	
DH-DSS-DES-CBC-SHA	SSLv3	DH/DSS	DH	DES(56)	SHA1	
DES-CBC-SHA–SSL3	SSLv3	RSA	RSA	DES(56)	SHA1	
DES-CBC-SHA–SSL2	SSLv2	RSA	RSA	DES(64)	SHA1	
DES-CBC-MD5	SSLv2	RSA	RSA	DES(56)	MD5	
IDEA-CBC-SHA	SSLv3	RSA	RSA	IDEA(128)	SHA1	
IDEA-CBC-MD5	SSLv2	RSA	RSA	IDEA(128)	MD5	
NULL-MD5–SSL3	SSLv3	RSA	RSA	None	MD5	
NULL-MD5–SSL2	SSLv2	RSA	RSA	None	MD5	
RC2-CBC-MD5	SSLv2	RSA	RSA	RC2(128)	MD5	
EXP-RC2-CBC-MD5	SSLv3	RSA(512)	RSA	RC2(40)	MD5	☑
EXP-RC2-CBC-MD5	SSLv2	RSA(512)	RSA	RC2(40)	MD5	☑
EXP1024-RC2-CBC-MD5	TLS1	RSA(1024)	RSA	RC2(56)	MD5	☑
AECDH-RC4-SHA	TLS1	ECDH	None	RC4(128)	SHA1	
ECDHE-RSA-RC4-SHA	TLS1	ECDH	RSA	RC4(128)	SHA1	
ECDH-RSA-RC4-SHA	TLS1	ECDH	RSA	RC4(128)	SHA1	
ECDHE-ECDSA-RC4-SHA	TLS1	ECDH	ECDSA	RC4(128)	SHA1	

Name	Mode	Key Ex.	Auth.	Encryption method (key length)	Message digest algorithm	Export
ECDH-ECDSA-RC4-SHA	TLS1	ECDH	ECDSA	RC4(128)	SHA1	
DHE-DSS-RC4-SHA	TLS1	DH	DSS	RC4(128)	SHA1	
ADH-RC4-MD5	SSLv3	DH	None	RC4(128)	MD5	
RC4-SHA	SSLv3	RSA	RSA	RC4(128)	SHA1	
RC4-MD5–SSL3	SSLv3	RSA	RSA	RC4(128)	MD5	
RC4-MD5–SSL2	SSLv2	RSA	RSA	RC4(128)	MD5	
RC4-MD5	SSLv2	RSA	RSA	RC4(128)	MD5	
EXP-ADH-RC4-MD5	SSLv3	DH(512)	None	RC4(40)	MD5	☑
EXP-RC4-MD5	SSLv3	RSA(512)	RSA	RC4(40)	MD5	☑
EXP-RC4-MD5	SSLv2	RSA(512)	RSA	RC4(40)	MD5	☑
EXP1024-DHE-DSS-RC4-SHA	TLS1	DH(1024)	DSS	RC4(56)	SHA1	☑
EXP1024-RC4-SHA	TLS1	RSA(1024)	RSA	RC4(56)	SHA1	☑
EXP1024-RC4-MD5	TLS1	RSA(1024)	RSA	RC4(56)	MD5	☑
RC4-64-MD5	SSLv2	RSA	RSA	RC4(64)	MD5	
KRB5–DES–CBC–SHA	SSLv3	KRB5	KRB5	DES(64)	SHA1	
KRB5-DES-CBC3-SHA	SSLv3	KRB5	KRB5	DES(192)	SHA1	
KRB5-RC4-SHA	SSLv3	KRB5	KRB5	RC4(128)	SHA1	
KRB5-IDEA-CBC-SHA	SSLv3	KRB5	KRB5	IDEA(128)	SHA1	
KRB5-DES-CBC-MD5	SSLv3	KRB5	KRB5	DES(64)	MD5	
KRB5-DES-CBC3-MD5	SSLv3	KRB5	KRB5	DES(192)	MD5	
KRB5-RC4-MD5	SSLv3	KRB5	KRB5	RC4(128)	MD5	
KRB5-IDEA-CBC-MD5	SSLv3	KRB5	KRB5	IDEA(128)	MD5	
EXP-KRB5-DES-CBC-SHA	SSLv3	KRB5	KRB5	DES(40)	SHA1	☑
EXP-KRB5-RC2-CBC-SHA	SSLv3	KRB5	KRB5	RC2(40)	SHA1	☑
EXP-KRB5-RC4-SHA	SSLv3	KRB5	KRB5	RC4(40)	SHA1	☑
EXP-KRB5-DES-CBC-MD5	SSLv3	KRB5	KRB5	DES(40)	MD5	☑
EXP-KRB5-RC2-CBC-MD5	SSLv3	KRB5	KRB5	RC2(40)	MD5	☑
EXP-KRB5-RC4-MD5	SSLv3	KRB5	KRB5	RC4(40)	MD5	☑

The following table contains abbreviations used in Table 9.

Table 10: SSL Cipher Abbreviations

Abbreviation	Description
3DES	Triple Data Encryption Standard. 3DES is a mode of the DES encryption algorithm that encrypts data three times.
ADH	Anonymous Diffie Hellman: The base Diffie-Hellman algorithm is used, but with no authentication.
AES	Advanced Encryption Standard.
CBC	Cipher Block Chaining encryption mode.
DES	Data Encryption Standard (DES) is a cipher (a method for encrypting information).
DH	Diffie-Hellman key exchange is a cryptographic protocol that enables two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.
DHE	Ephemeral Diffie-Hellman key exchange that creates ephemeral (one-time) secret keys. This is possibly the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.
DSS	Digital Signature Standard. DSS is a United States Federal Government standard for digital signatures.
ECDH	Elliptic Curve Diffie-Hellman is a key agreement protocol that enables two parties to establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. ECDH is a variant of the Diffie-Hellman protocol using elliptic curve cryptography.
ECDSA	Elliptic Curve Digital Signature Algorithm.
Fortezza	Fortezza is an information security system developed by the United States Federal Government. Fortezza PC Card security tokens contain an NSA-approved security microprocessor called Capstone (MYK-80) that implements the Skipjack encryption algorithm.
GCM	Galois/Counter Mode for symmetric key cryptographic block ciphers. It combines the well-known counter mode of encryption with the new Galois mode of authentication.
IDEA	International Data Encryption Algorithm (IDEA). IDEA is a block cipher.
MD5	Message-Digest algorithm 5.
RC2	Block cipher with a variable size key.

Abbreviation	Description
RC4	Software stream cipher (also known as ARC4 or ARCFOUR).
RSA	An algorithm for public-key encryption.
SHA1	Secure Hash Algorithm 1.
SRP	Secure Remote Password, a cryptographically strong network authentication mechanism

Appendix B

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
- XML Parser Interface

The tables in this appendix list the properties and methods of the interfaces supported by WebLOAD.

XML Document Interface Properties

Table 11: XML Document Interface Properties

Property	Description
doctype	A read-only property that gets the node for the DTD specified for the document. If no DTD was specified, null is returned.
documentElement	A read/write property that gets/sets the root node of the document.
implementation	A read-only property that returns the implementation interface for this document.
parseError	A read-only property that provides an object that summarizes the last parsing error encountered.

Property	Description
preserveWhitespace	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 false.
readyState	A read-only property indicating the status of instantiating the XML processor and document download. The value of the readyState property is summarized in the table.
resolveExternals	A read-write property that informs the parser that resolvable namespaces (a namespaces URI that begin with an "`x-schema:`" prefix), DTD external subsets, and external entity references should be resolved at parse time.
url	A read-only property that returns the canonicalized URL for the XML document specified in the last call to `load()`.
validateOnParse	A read/write property that turns validation on at parse time if the value of `bool` is true, off if `validate` is false.

XML Document Interface Methods

Table 12: XML Document Interface Methods

Method	Description
abort()	Aborts an asynchronous download in progress.
createAttribute(name)	Creates a node of type ATTRIBUTE with the name supplied.
CreateCDATASection (data)	Creates a node of type CDATA_SECTION with nodeValue set to data.
createComment(data)	Creates a node of type COMMENT with nodeValue set to data.
createDocumentFragment	Creates a node of type DOCUMENT_FRAGMENT in the context of the current document.
createElement(tagName)	Creates a node of type ELEMENT with the nodeName of tagName.
createEntityReference(name)	Creates a node of type ENTITY_REFERENCE where name is the name of the entity referenced.
CreateNode(type, name, namespaceURI)	Creates a node of the type specified in the context of the current document. Allows nodes to be created as a specified namespace.

Method	Description
CreateProcessing Instruction (target, data)	Creates a node of type PROCESSING_INSTRUCTION with the target specified and nodeValue set to data.
createTextNode(data)	Creates a node of type TEXT with nodeValue set to data.
GetElementsByTagName (tagname)	Returns a collection of all descendent Element nodes with a given tagName.
load(url)	Loads an XML document from the location specified by the `url`. If the `url` cannot be resolved or accessed or does not reference an XML document, the documentElement is set to null and an error is returned. Returns a Boolean.
loadXML(xmlstring)	Loads an XML document using the supplied string. xmlstring can be an entire XML document or a well-formed fragment. If the XML within xmlstring cannot be loaded, the documentElement is set to null and an error is returned.
NodeFromID(idstring)	Returns the node that has an ID attribute with the value corresponding to idString.
Save()	Serialize the XML. The parameter can be a filename, an ASP response, an XML Document, or any other COM object that supports Istream, IpersistStream, or IpersistStreamInit.

Node Interface Properties

Table 13: Node Interface Properties

Property	Description
attributes	A read-only property that returns a `NamedNodeMap` containing attributes for this node.
BaseName	A read-only property that returns the right-hand side of a namespace qualified name. For example, `yyy` for the element `<xxx:yyy>`. `BaseName` must always return a non-empty string.
childNode	A read-only property that returns a `NodeList` containing all children of the node.
DataType	A read-write property that indicates the node type.

Property	Description
Definition	A read-only property whose value is the node that contains the definition for this node.
FirstChild	A read-only property that returns the first child node. If the node has no children, `firstChild` returns null.
LastChild	A read-only property that returns the last child node. If the node has no children, `lastChild` returns null.
NextSibling	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.
NamespaceURI	A read-only property that returns the URI for the namespace (the `uuu` portion of the namespace declaration `xmlns:nnn="uuu"`). If there is no namespace on the node that is defined within the context of the document, "" is returned.
NodeName	A read-only property indicating the name of the node.
NodeType	A read-only property indicating the type of node.
NodeTypeString	Returns the node type in string form.
NodeTypedValue	A read/write property for the typed value of the node.
NodeValue	A read/write property for the value of the node.
OwnerDocument	A property that indicates the document to which the node belongs or when the node is removed from a document.
parentNode	A read-only property that provides a pointer to the parent.
parsed	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.
prefix	A read-only property that returns the prefix specified on the element, attribute of entity reference. For example, `xxx` for the element <xxx:yyy>. If there is no prefix specified, "" is returned.

Property	Description
previousSibling	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.
specified	A read-only property indicating the node was specified directly in the XML source and not implied by the DTD schema.

text	A string representing the content of the element and all descendents. For example "`content of tag`" in
	`<sometag size=34>`
	`  content of tag`
	`</sometag>`.

xml	A read-only property that returns the XML representation of the node and all its descendants as a string.

Node Interface Methods

Table 14: Node Interface Methods

Method	Description
appendChild(newChild)	A method to append `newChild` as the last child of this node.
cloneNode(deep)	A method to create a new node that is an exact clone (same name, same attributes) as this node. When deep is false, only the node and attributes without its children are cloned. When deep is true, the node and all its descendants are cloned.
hasChildNodes()	A method that indicates whether the node has children.
InsertBefore (newChild, oldChild)	A method to insert `newChild` as a child of this node. `oldChild` is returned. `oldNode` must be a child node of the element, otherwise an error is returned. If `newChild` is null, the `oldChild` is removed.
removeChild(child)	A method to remove a `childNode` from a node. If `childNode` is not a child of the node, an error is returned.

Method	Description
ReplaceChild (newChild, oldChild)	A method to replace `oldChild` with `newChild` as a child of this node.
selectNodes(query)	Returns a `NodeList` containing the results of the query indicated by query, using the current node as the query context. If no nodes match the query, an empty `NodeList` is returned. If there is an error in the query string, the DOM error reporting is used.
SelectSingleNode (query)	Returns a single node that is the first node in the `NodeList` 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.
TransformNode (stylesheetDOMNode)	Returns the results of processing the source `DOMNode` and its children with the stylesheet indicated by `stylesheetDOMNode`. 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 xsl namespace, in which case this node is treated as a standalone.
TransformNodeToObject (stylesheet, Object)	Sends the results of the transform to the requested object, either in `IStream` or a DOM Document.

Node List Interface

Table 15: Node List Interface

Property	Description
length	The number of nodes in the `NodeList`. The length of the list will change dynamically as children or attributes are added/deleted from the element.
nextNode	Returns the next node in the `NodeList` based on the current node.

Method	Description
item(index)	Returns the node in the `NodeList` with the specified index.

Method	Description
reset()	Returns the iterator to the uninstantiated state; that is, before the first node in the `NodeList`.

NamedNodeMap Interface

Table 16: NamedNodeMap Interface

Property	Description
length	The number of nodes in the `NamedNodeMap`. The length of the list will change dynamically as children or attributes are added/deleted from the element.

Method	Description
getNamedItem(name)	Returns the node corresponding to the attribute with name. If name is not an attribute, null is returned.
GetQualifiedItem (baseName, namespaceURI)	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.
item(index)	Returns the node in the `NameNodeMap` 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.
nextnode()	Returns the next node in the `NodeList` based on the current node.
RemovedNamedItem (name)	Removes the attribute node corresponding to name and returns the node. If name is not an attribute, null is returned.
RemoveQualifiedItem (basename, namespaceURI)	Removes the `nameSpaceURI` qualified attribute node corresponding to `baseName` and returns the node. If the qualified name is not an attribute, null is returned.
reset()	Returns the iterator to the uninstantiated state; that is before the first node in the `NodeList`.

Method	Description
SetNamedItem (namedItem)	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.

ParseError Interface

Table 17: ParseError Interface

Item	Description
errorcode	Returns the error code number in decimal.
filepos	Returns the absolute file position where the error occurred.
line	Returns number of the line containing the error.
linepos	Returns the character position where the error occurred.
reason	Returns the reason for the error.
srcText	Returns the full text of the line containing the error.
url	Returns the URL of the XML file containing the error.

Implementation Interface

Table 18: Implementation Interface

Item	Description
HasFeature (feature, version)	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.

Appendix C

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*)

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 19: 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	The client has requested, via the `Upgrade` 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 `Upgrade` header field immediately after the empty line which terminates this `101` response. 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.

Success 2XX

The *2XX* class of status code indicates that the client's request was successfully received, understood, and accepted.

Table 20: Successful 2XX Message Set

Message	Description
200 OK	The request has succeeded. The information returned with a `200` response is dependent on the method used in the request. For example: `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

Message	Description
201 _Created	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 `Location` header field. 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 `Content-Type` 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. A `201` response may contain an `ETag` response header field indicating the current value of the entity tag for the requested variant just created.
202 Accepted	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. 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.
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`).

Message	Description
204 No Content	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. If the client is a user agent, it *should not change its document view* 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. The `204` response must not include a message-body, and thus is always terminated by the first empty line after the header fields.
205 Reset Content	The server has fulfilled the request and the user agent *should reset the document view* 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.

Message	Description
206 Partial Content	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. The response must include one of the following header fields: • `Content-Range` header field indicating the range included with this response. • 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 • `Expires`, `Cache-Control`, and/or `Vary`, if the field-value might differ from that sent in any previous response for the same variant. 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. A cache must not combine a `206` response with other previously cached content if the `ETag` or `Last-Modified` headers do not match exactly. A cache that does not support `Range` and `Content-Range` headers must not cache `206` (`Partial`) responses.

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 21: Redirectional 3XX Message Set

Message	Description
300 Multiple Choices	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. Unless it was a `HEAD` 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 `Content-Type` 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. If the server has a preferred choice of representation, it should include the specific URI for that representation in the `Location` field. User agents may use the `Location` field value for automatic redirection. This response is cacheable unless otherwise indicated.
301 Moved Permanently	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 `Request-URI` to one or more of the new references returned by the server, where possible. This response is cacheable unless otherwise indicated. The new permanent URI should be identified by the `Location` field in the response. Unless the request method was `HEAD`, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s). If the `301` status code is received in response to a request other than `GET` or `HEAD`, 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. **Note:** When automatically redirecting a `POST` request after receiving a `301` status code, some existing HTTP/1.0 user agents will erroneously change it into a `GET` request.

Message	Description
302 Found	The requested resource *temporarily* resides under a different URI. Since the redirection might be altered on occasion, the client should continue to use the `Request-URI` for future requests. This response is only cacheable if indicated by a `Cache-Control` or `Expires` header field. The temporary URI should be identified by the `Location` field in the response. Unless the request method was `HEAD`, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s). If the `302` status code is received in response to a request other than `GET` or `HEAD`, 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. **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 `302` as if it were a `303` response, performing a `GET` on the `Location` field-value regardless of the original request method. The status codes `303` and `307` have been added for servers that wish to make unambiguously clear which kind of reaction is expected of the client.
303 See Other	The response to the request can be found under a different URI and should be retrieved using a `GET` method on that resource. This method exists primarily to allow the output of a `POST`-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 `303` response must not be cached, but the response to the second (redirected) request might be cacheable. The different URI should be identified by the `Location` field in the response. Unless the request method was `HEAD`, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s). **Note:** Many pre-HTTP/1.1 user agents do not understand the `303` status. When interoperability with such clients is a concern, the `302` status code may be used instead, since most user agents react to a `302` response as described here for `303`.

Message	Description
304 Not Modified	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. The response must include the following header fields: • `Date`, unless its omission is required. f 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. 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 *must not* include other entity-headers. This prevents inconsistencies between cached entity-bodies and updated headers. If a `304` response indicates an entity not currently cached, then the cache must disregard the response and repeat the request without the conditional. 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.
305 Use Proxy	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. **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.
306 (Unused)	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.

Message	Description
307 Temporary Redirect	The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client should continue to use the `Request-URI` for future requests. This response is only cacheable if indicated by a `Cache-Control` or `Expires` header field. The temporary URI should be identified by the `Location` field in the response. Unless the request method was `HEAD`, 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. If the `307` status code is received in response to a request other than `GET` or `HEAD`, 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.

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 22: 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.

Message	Description
401 Unauthorized	The request requires user authentication. The response must include a `WWW-Authenticate` header field containing a `challenge` applicable to the requested resource. The client may repeat the request with a suitable `Authorization` header field. If the request already included Authorization 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 `HEAD` 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` (`Not Found`) can be used instead.
404 Not Found	The server has not found anything matching the `Request-URI`. No indication is given of whether the condition is temporary or permanent. The `410` (`Gone`) 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 `Request-Line` is not allowed for the resource identified by the `Request-URI`. The response must include an `Allow` header containing a list of valid methods for the requested resource.

Message	Description
406 Not Acceptable	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. Unless it was a `HEAD` 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 `Content-Type` 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.  **Note:** HTTP/1.1 servers are allowed to return responses which are not acceptable according to the Accept Headers 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.
407 Proxy Authentication Required	This code is similar to `401` (`Unauthorized`), but indicates that the client must first authenticate itself with the proxy. The proxy must return a `Proxy-Authenticate` header field containing a challenge applicable to the proxy for the requested resource. The client may repeat the request with a suitable `Proxy-Authorization` 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.
409 Conflict	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. Conflicts are most likely to occur in response to a `PUT` request. For example, if versioning were being used and the entity being `PUT` 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 `Content-Type`.

Message	Description
410 Gone	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 `Request-URI` 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` (`Not Found`) should be used instead. This response is cacheable unless indicated otherwise. 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.
411 Length Required	The server refuses to accept the request without a defined `Content-Length`. The client may repeat the request if it adds a valid `Content-Length` header field containing the length of the message-body in the request message.
412 Precondition Failed	The precondition given in one or more of the request-header fields evaluated to `false` 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.
413 Request Entity Too Large	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. If the condition is temporary, the server should include a `Retry-After` header field to indicate that it is temporary and after what time period has elapsed may the client try again.
414 Request-URI Too Long	The server is refusing to service the request because the `Request-URI` is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a `POST` request to a `GET` 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 `Request-URI`.
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.

Message	Description
416 Requested Range Not Satisfiable	A server should return a response with this status code if: • A request included a `Range` request-header field. • None of the range-specifier values in this field overlap the current extent of the selected resource. • The request did not include an `If-Range` request-header field. For byte-ranges, this means that the `first-byte-pos` of all of the `byte-range-spec` values were greater than the current length of the selected resource. When this status code is returned for a byte-range request, the response should include a `Content-Range` entity-header field specifying the current length of the selected resource. This response must not use the `multipart/byteranges` content-type.
417 Expectation Failed	The expectation identified in an `Expect` 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.

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 23: 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.

Message	Description
503 Service Unavailable	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 `Retry-After` header. If no `Retry-After` is given, the client should handle the response as it would for a `500` response.  **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.
504 Gateway Timeout	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.  **Note:** Implementers should note that some deployed proxies are known to return `400` or `500` when DNS lookups time out.
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.

Appendix D

WebLOAD–supported Character Sets

WebLOAD supports the following character sets for use with character encoding functions.

Table 24. Supported Character Sets

Character Set	Value
Default	0
Arabic (864)	864
Arabic (ASMO 708)	708
Arabic (DOS)	720
Arabic (ISO 8859-6)	28596
Arabic (Mac)	10004
Arabic (Windows)	1256
Baltic (ISO 8859-4)	28594
Baltic (Windows)	1257
Baltic / Estonian (ISO 8859-13)	28603
Celtic (ISO 8859-14)	28604
Central European (DOS)	852
Central European (ISO 8859-2)	28592
Central European (Mac)	10029
Central European (Windows)	1250
Chinese Simplified (GB18030)	54936
Chinese Simplified (GB2312)	936
Chinese Simplified (GB2312-80)	20936
Chinese Simplified (HZ)	52936
Chinese Simplified (ISO 2022)	50227
Chinese Simplified (Mac)	10008
Chinese Traditional (Big5)	950

Character Set	Value
Chinese Traditional (EUC)	51950
Croatian (Mac)	10082
Cyrillic (DOS)	866
Cyrillic (ISO 8859-5)	28595
Cyrillic (KOI8-R)	20866
Cyrillic (KOI8-U)	21866
Cyrillic (Mac)	10007
Cyrillic (primarily Russian)	855
Cyrillic (Windows)	1251
Greek (ISO 8859-7)	28597
Greek (Mac)	10006
Greek (Windows)	1253
Hebrew (DOS)	862
Hebrew (ISO-Logical)	38598
Hebrew (ISO-Visual)	28598
Hebrew (Mac)	10005
Hebrew (Windows)	1255
Icelandic (Mac)	10079
Japanese (EUC)	51932
Japanese (JIS)	50220
Japanese (Shift-JIS)	932
Korean (EUC)	51949
Korean (ISO 2022)	50225
Korean (Johab)	1361
Korean (Unified Hangul Code)	949
Nordic (ISO 8859-10)	28600
Romanian (ISO 8859-16)	28606
Romanian (Mac)	10010
South European (ISO 8859-3)	28593
Tahi (ISO 8859-11)	28601
Thai (Windows)	874
Turkish (DOS)	857

Character Set	Value
Turkish (ISO 8859-9)	28599
Turkish (Mac)	10081
Turkish (Windows)	1254
Ukrainian (Mac)	10017
Unicode (UTF-7)	65000
Unicode (UTF-8)	65001
Unicode UTF-16, big endian	1201
Unicode UTF-16, little endian	1200
Unicode UTF-32, big endian	12001
Unicode UTF-32, little endian	12000
Vietnamese (Windows)	1258
Western (ISO-8859-15)	28605
Western European (DOS)	850
Western European (ISO 8859-1)	28591
Western European (Mac)	10000
Western European (Windows)	1252

Appendix E

Glossary

Glossary Terms

Glossary Term	Description
AAT	An older, obsolete WebLOAD utility that was used for recording web session activities as a JavaScript file. It is replaced by WebLOAD Recorder.
Aborted Rounds	The number of times the Virtual Clients started to run a script but did not complete the script, during the last reporting interval. This might be due to a session being stopped either automatically or manually by the user.
script	Specification of the sequence of HTTP protocol calls sent by Virtual Clients to the SUT (System Under Test). Scripts are written in JavaScript. You can either write scripts as a text file or generate them automatically using the WebLOAD Recorder.
Application Being Tested (ABT)	See *SUT*.
Attempted Connections	The total number of times the Virtual Clients attempted to connect to the SUT during the last reporting interval.
Automatic Transaction counters	If you have Automatic Transactions enabled, WebLOAD creates three counters for each GET and POST statement in the script: • The total number of times it occurred • The number of times it succeeded • The number of times it failed during the last reporting interval.
Average	For timers, average is the total amount of time counted by the timer (not the elapsed time) divided by the Count (that is, the total number of readings). For example, the average for Transaction Time is the amount of time it took to complete all the successful transactions, divided by the number of successful transactions (the Count).

Glossary Term	Description
Built-in Timer	A timer measures the time required to perform a given task. WebLOAD supports both programmed timers and built-in timers. ROUND TIME is a built-in timer. The ROUND TIME is the time needed for one complete execution of a script.
Connect Time	The time it takes for a Virtual Client to connect to the System Under Test (the SUT), in seconds. In other words, the time it takes from the beginning of the HTTP request to the TCP/IP connection. The value posted in the Current Value column is the average time it took a Virtual Client to connect to the SUT during the last reporting interval. If the Persistent Connection option is enabled, there may not be a value for Connect Time because the HTTP connection remains open between successive HTTP requests.
Connection Speed (Bits Per Second)	The number of bits transmitted back and forth between the Virtual Clients and the System Under Test (SUT), divided by the time it took to transmit those bits, in seconds. You can set the Virtual Clients to emulate a particular connection speed during the test, either by using the Variable Connection Speed settings, or by coding the connection speed in the script. If a connection speed is specified for the test, WebLOAD reports it in the Statistics Report. The value posted in the Current Value column is the number (sum) of bits passed per second during the last reporting interval. It should match, very closely, the connection speed you specified for the test.

Glossary Term	Description
Console	The WebLOAD component that manages the test session. The Console performs the following: • Configures Load Session hosts and scripts • Schedules Load Session scripts • Configures Goal–Oriented test sessions • Monitors the application's performance under the generated load • Manages the Load Session as it is running, allowing you to pause, stop, and continue Load Session components as needed • Displays the current performance of the SUT • Provides a final performance reports for Probing Clients and Virtual Clients • Manages exporting of performance reports
Count	(For timers only.) The total number of readings (the number of times the item being timed has occurred) for the timed statistic since the beginning of the test. For example, for Transaction Time, Count shows the number of transactions that have been completed.
Current Slice	The value posted for this reporting interval in the Statistics Report main window.
Current Slice Average	For per time unit statistics and counters, average is the total of all of the current values for the last reporting interval, divided by the number of readings. For timers, average is the total amount of time counted by the timer (not the elapsed time), divided by the Count (that is, the total number of readings for the last reporting interval). For example, the average for Transaction Time is the amount of time it took to complete all the successful transactions in the last reporting interval, divided by the number of successful transactions (the Current Slice Count).
Current Slice Count	(For timers only.) The total number of readings (the number of times the item being timed has occurred) for the timed statistic for the last reporting interval. For example, for Transaction Time, Current Slice Count shows the number of transactions that have been completed over the last reporting interval.
Current Slice Max	The highest value reported for this statistic over the last reporting interval.
Current Slice Min	The lowest value reported for this statistic over the last reporting interval.

Glossary Term	Description
Current Slice Standard Deviation	The average amount the measurement for this statistic varies from the average over the last reporting interval.
Current Slice Sum	The aggregate or total value for this statistic in this script over the last reporting interval.
DNS Lookup Time	The time it takes to resolve the host name and convert it to an IP address by calling the DNS server.
Failed Connections	The total number of times the Virtual Clients tried to connect to the SUT but were unsuccessful, during the last reporting interval. This number is always less than or equal to the number of failed hits because hits can fail for reasons other than a failed connection.
Failed Hits	The total number of times the Virtual Clients made an HTTP request but did not receive the correct HTTP response from the SUT during the last reporting interval. Note that each request for each `gif`, `jpeg`, `html` file, etc., is a single hit.
Failed Hits Per Second	The number of times the Virtual Clients did not obtain the correct HTTP response, divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of unsuccessful HTTP requests per second during the last reporting interval.
Failed Pages Per Second	The number of times the Virtual Clients did not obtain the correct response to an upper level request, divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of unsuccessful requests per second during the last reporting interval.
Failed Rounds	The total number of times the Virtual Clients started but did not complete the script during the last reporting interval.
Failed Rounds Per Second	The number of times the Virtual Clients started but did not complete an iteration of the script, divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of failed iterations of the script per second during the last reporting interval.
First Byte	The time it takes a Virtual Client to receive the first byte of data.
Gallery	See *Templates Gallery*.

Glossary Term	Description
Goal–Oriented Test	A WebLOAD component enabling you to define the performance goals required, and view the status of your application when it is operating under this performance goal. WebLOAD provides a Goal–Oriented Test Wizard for configuring these performance goals. WebLOAD automatically accelerates the number of Virtual Clients accessing your website until you meet your performance goal.  **Note:** The Goal-Oriented Test Wizard was previously called the Cruise Control Wizard.
Goal–Oriented Test Wizard	See *Goal–Oriented Test*.
Hit Time	The time it takes to complete a successful HTTP request, in seconds. Each request for each `gif`, `jpeg`, `html` file, etc., is a single hit. The time of a hit is the sum of the Connect Time, Send Time, Response Time, and Process Time. The value posted in the Current Value column is the average time it took to make an HTTP request and process its response during the last reporting interval.
Hits	The total number of times the Virtual Clients made HTTP requests to the System Under Test (SUT) during the last reporting interval. For example, a Get statement for a URL retrieves a page. The page can include any number of graphics and contents files. Each request for each `gif`, `jpeg`, `html` file, etc., is a single hit.
Hits Per Second	The number of times the Virtual Clients made an HTTP request, divided by the elapsed time, in seconds. Each request for each `gif`, `jpeg`, `html` file, etc. is a single hit. The value posted in the Current Value column is the number (sum) of HTTP requests per second during the last reporting interval.
Host	A computer connected via a network, participating in a test session. Each Host in a test session has assigned tasks. A host can act as either a Load Machine or a Probing Client Machine. All hosts participating in a test session must be accessible to the Console over a network. Therefore they must run TestTalk, the network agent.

Glossary Term	Description
HTTP Response Status	WebLOAD creates a row in the Statistics Report for each kind of HTTP status code it receives as an HTTP response from the SUT (redirection codes, success codes, server error codes, or client error codes). The value posted is the number of times the Virtual Clients received that status code during the last reporting interval.
Integrated Report	A single configurable report that can integrate both standard performance data, and data from the NT Performance Monitor. This report gives you a more complete picture of the performance of your application. The data to be monitored and the data to be displayed in the report are both configurable in the Console.
Internet Productivity Pack (IPP)	Provides a set of protocol implementations enabling you to load-test your application using these protocols.
Java and ActiveX counters	You can add function calls to your scripts that enable you to instantiate and call methods and properties in Java and ActiveX components (see the *WebLOAD Scripting Guide*). If there are ActiveX or Java function calls in the script that you are running, WebLOAD reports three counters for them in the Statistics Report: • The total number of times it occurred • The number of times it succeeded • The number of times it failed during the last reporting interval. The row heading in the Statistics Report is the name of the function call.
Java and ActiveX timers	You can add function calls to your scripts that enable you to instantiate and call methods and properties in Java and ActiveX components (see the *WebLOAD Scripting Guide*). If there are ActiveX or Java function calls in the script you are running, WebLOAD reports timers for them in the Statistics Report. The timer value is the average amount of time it took to complete the function call, in seconds, during the last reporting interval. The row heading in the Statistics Report is the name of the function call.

Glossary Term	Description
Load Generator	The component of the Load Machine that generates Virtual Clients. Load Generators have the task of bombarding the System Under Test with HTTP protocol call requests as defined in the script. WebLOAD assesses the application's performance by measuring the response time experienced by the Virtual Clients. The number of Virtual Clients at any given moment is determined by the user.
Load Generator Machine	See *Load Machine*.
Load Machine	A host that runs Load Generators. Load Generators bombard the application under test with a large load, to enable complete scalability and integrity testing.
Load Session	A Load Session includes both the complete Load Template and the results obtained while running that Load Session. A Load Template consists of information about the hosts and scripts participating in the current Load Session. The Load Template will also include scheduling information. The complete Load Template is illustrated in the Session Tree. Storing a Load Template saves you time when repeatedly running WebLOAD with the same, or even a similar network configuration, since you don't have to recreate your Load Template from scratch each time you want to start working. Storing Load Session results can be useful when you want to examine results from multiple test sessions or for analyzing test session results.
Load Size	The number of Virtual Clients running during the last reporting interval.
Load Template	A Load Template contains the complete Load Session definition, without the test results. A Load Template includes information about the participating hosts and the scripts used in the current Load Session. The definition also includes scheduling information and the configuration of the Server Monitor and Integrated Reports. The complete Load Template is illustrated in the Session Tree. Storing a Load Template saves you time when repeatedly running WebLOAD with the same, or even a similar network configuration, since you do not have to recreate your Load Template from scratch each time you rerun a test.

Glossary Term	Description
Page Time	The time it takes to complete a successful upper level request, in seconds. The Page Time is the sum of the Connection Time, Send Time, Response Time, and Process Time for all the hits on a page. The value posted in the Current Value column is the average time it took the Virtual Clients to make an upper level request and process its response during the last reporting interval.
Pages	The total number of times the Virtual Client made upper level requests, both successful and unsuccessful, during the last reporting interval.
Pages Per Second	The number of times the Virtual Clients made upper level requests divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of requests per second during the last reporting interval.
Per Time Unit statistics	Ratios that calculate an average value for an action or process. For example: Transactions Per Second, Rounds Per Second.
Portfolio	A Portfolio of reports enables you to generate a single, inclusive report that contains all the charts generated by the templates included in the portfolio.
Probing Client	A software program which "bombards" the SUT as a single Virtual Client, to further measure the performance of the SUT. WebLOAD generates exact values for Probing Client performance.
Probing Client Machines	Hosts running Probing Client software simulating one Virtual Client, and running at the same time as Load Machines.
Probing Client software	See *Probing Client*.
Process Time	The time it takes WebLOAD to parse an HTTP response from the SUT and then populate the document-object model (DOM), in seconds. The value posted in the Current Value column is the average time it took WebLOAD to parse an HTTP response during the last reporting interval.
Receive Time	The elapsed time between receiving the first byte and the last byte.
Report Portfolio	See *Portfolio*.

Glossary Term	Description
Resource Manager	Distributes and circulates WebLOAD testing resources (Virtual Clients and Probing Clients) amongst users on a "need to use" basis. The Resource Manager is packaged with a maximum number of Virtual Clients, Probing Clients and Connected Workstation ports, as defined by the WebLOAD package. With the Resource Manager, every WebLOAD Console can operate in Standalone Workstation mode or Connected Workstation mode.
Response Data Size	The size, in bytes, of all the HTTP responses sent by the SUT during the last reporting interval. WebLOAD uses this value to calculate Throughput (bytes per second).
Response Time	The time it takes the SUT to send the object of an HTTP request back to a Virtual Client, in seconds. In other words, the time from the end of the HTTP request until the Virtual Client has received the complete item it requested. The value posted in the Current Value column is the average time it took the SUT to respond to an HTTP request during the last reporting interval.
Responses	The number of times the SUT responded to an HTTP request during the last reporting interval. This number should match the number of successful hits.
Round Time	The time it takes one Virtual Client to finish one complete iteration of a script, in seconds. The value posted in the Current Value column is the average time it took the Virtual Clients to finish one complete iteration of the script during the last reporting interval.
Rounds	The total number of times the Virtual Clients attempted to run the script during the last reporting interval.
Rounds Per Second	The number of times the Virtual Clients attempted to run the script, divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of attempts (both successful and unsuccessful) per second during the last reporting interval.

Glossary Term	Description
Send Time	The time it takes the Virtual Client to write an HTTP request to the SUT, in seconds. The value posted in the Current Value column is the average time it took the Virtual Clients to write a request to the SUT during the last reporting interval.
Server Performance Measurements	If you selected Performance Monitor statistics for the report, WebLOAD creates a row for them and reports their values in the Statistics Report. For definitions of the statistics, see the Server Monitor Definition dialog box. Be selective when choosing server performance measurements , otherwise the system resources required to manage the data might affect the Console.
Session Tree	A graphic representation of a Load Template and status. It illustrates the different components of a test session, including Load Machines and Probing Clients, the scripts that they execute, and their status.
Single Client	See *Probing Client*.
Standard Deviation	The average amount the measurement varies from the average since the beginning of the test.
Successful Connections	The total number of times the Virtual Clients were able to successfully connect to the SUT during the last reporting interval. This number is always less than or equal to the number of successful hits because several hits might use the same HTTP connection if the Persistent Connection option is enabled.
Successful Hits	The total number of times the Virtual Clients made an HTTP request and received the correct HTTP response from the SUT during the last reporting interval. Each request for each `gif`, `jpeg`, `html` file, etc., is a single hit.
Successful Hits Per Second	The number of times the Virtual Clients obtained the correct HTTP response to their HTTP requests divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of successful HTTP requests per second during the last reporting interval.
Successful Pages Per Second	The value posted in the Current Value column is the number (sum) of successful requests per second during the last reporting interval.

Glossary Term	Description
Successful Rounds	The total number of times the Virtual Clients completed one iteration of the script during the last reporting interval.
Successful Rounds Per Second	The number of times the Virtual Clients completed an entire iteration of the script, divided by the elapsed time, in seconds. The value posted in the Current Value column is the number (sum) of successful iterations of the script per second during the last reporting interval.
SUT	The system running the Web application currently under test. The SUT (System Under Test) is accessed by clients through its URL address. The SUT can reside on any machine or on multiple machines, anywhere on the global Internet or your local intranet.
Template	See *Load Template*.
Templates Gallery	The Templates Gallery is a single entity that contains predefined templates, user-defined templates, and portfolios.
Test Program	See *Test Script*.
Test Script	The script. This defines the test scenario used in your Load Session. Scripts are written in JavaScript.
Test Template	See *Load Template*.
TestTalk	The network agent. This program enables communication between the Console and the host computers participating in the test.
Throttle Control	A WebLOAD component that enables you to dynamically change the Load Size while a test session is in progress.
Throughput (Bytes Per Second)	The average number of bytes per second transmitted from the SUT to the Virtual Clients running the script during the last reporting interval. In other words, this is the amount of the Response Data Size, divided by the number of seconds in the reporting interval.
Time to First Byte	The time that elapsed since a request was sent until the Virtual Client received the first byte of data.

Glossary Term	Description
User-defined Automatic Data Collection	If you have Automatic Data Collection enabled, WebLOAD creates three counters for each GET and POST statement in the script: • The total number of times the Get and Post statements occurred • The number of times the statements succeeded • The number of times the statements failed during the last reporting interval.
User-defined counters	Your own counters that you can add to scripts using the `SendCounter()` and the `SendMeasurement()` functions (see the *WebLOAD Scripting Guide*). If there is a user-defined counter in the script that you are running, WebLOAD reports the counter's values in the Statistics Report. The row heading is the name (argument) of the counter. That is, the row heading is the string in parenthesis in the `SendCounter()` or `SendMeasurement()` function call. The value reported is the number of times the counter was incremented during the last reporting interval.
User-defined timer	Timers that you can add to scripts to keep track of the amount of time it takes to complete specific actions (see the *WebLOAD Scripting Guide*). If there are any timers in the scripts that you are running, WebLOAD reports their values in the Statistics Report. The row heading is the name (argument) of the timer. That is, the row heading is the string in parenthesis in the `SetTimer()` function call. The timer represents the time it takes to complete all the actions between the `SetTimer()` call and its corresponding `SendTimer()` call, in seconds. The value posted is the average time it took a Virtual Client to complete the actions between the pair of timer calls, in seconds, during the last reporting interval.

Glossary Term	Description
User-defined Transaction counters	Transaction functions that you can add to scripts for functional tests (see the *WebLOAD Scripting Guide*). If there is a user-defined transaction function in the script that you are running, WebLOAD reports three counters for it in the Statistics Report: • The total number of times the transaction occurred • The number of times a transaction succeeded • The number of times a transaction failed during the last reporting interval. The row heading is the name (argument) of the transaction. That is, the row heading is the string in parenthesis in the `BeginTransaction()` function call.
User-defined Transactions timers	A timer for user-defined transaction functions. If there is a user-defined transaction function in the script that you are running, WebLOAD reports a timer for it in the Statistics Report. The row heading is the name (argument) of the user-defined transaction. That is, the row heading is the string in parenthesis in the `BeginTransaction()` function call. The timer represents the average time it took to complete all the actions between the `BeginTransaction()` call and its corresponding `EndTransaction()` call, in seconds, during the last reporting interval.
Virtual Client	Artificial entities run by Load Generators. Each such entity is a perfect simulation of a real client accessing the System Under Test (SUT) through a Web browser. Virtual Clients generate HTTP calls that access the SUT. The Load Generators that run Virtual Clients can reside anywhere on the Internet or on your local intranet. Scripts are executed by all the Virtual Clients in parallel, achieving simultaneous access to the SUT. The size of the load on your SUT is determined by the number of Virtual Clients being generated. You may define as many Virtual Clients as needed, up to the maximum supported by your WebLOAD "package."
WebLOAD Analytics	WebLOAD Analytics enables you to analyze data, and create custom, informative reports after running a WebLOAD test session.
WebLOAD Console	See *Console*.
WebLOAD Integrated Development Environment (IDE)	An easy-to-use tool for recording, creating, and authoring protocol scripts for the WebLOAD environment.

Glossary Term	Description
WebLOAD Load Template	See *Load Template*.
WebLoad Session	See *Load Session*.
WebLOAD Wizard	A WebLOAD Wizard that steps you through the configuration process. Each screen of the WebLOAD Wizard contains text explaining the configuration process. The WebLOAD Wizard enables you to create a basic Load Template. After using the demo, you can use the Console menus to add functionality not available through the WebLOAD Wizard.
WebRM	See *Resource Manager*.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X