



Radview Software How to

Performance Testing for Ajax Applications

Rich internet applications are growing rapidly and AJAX technologies serve as the building blocks for such applications. These new technologies bring new challenges to the performance tester.

In this document you will learn how to use WebLOAD to load test your Ajax applications. We will cover what Ajax is and how to work with different Ajax data types including XML, JSON and text-based.

Table of Contents

Overview.....	3
What is Ajax?	3
JSON.....	3
Testing Ajax applications	4
Configuration of recording options.....	4
Parsing responses from Ajax applications.....	4
Setting the SaveSource option	4
Working with different data types	5
Working with Get request instead of Post request	5
Recording the session.....	6
Debugging an Ajax enabled agenda script.....	6
The complete sample.....	6
Troubleshooting	7
Setting content type options	7
Future enhancements.....	7
Additional resources.....	8

Overview

Rich Internet applications (RIA) are web applications that have the features and functionality of traditional desktop applications. RIAs typically delegate the processing necessary for the user interface to the web client but keep the bulk of the data (i.e. maintaining the state of the program, the data etc) back on the application server. There are a number of useful technologies for building a good RIA, but the one we get asked about most frequently is Ajax. This document provides information on how to use WebLOAD (version 7.6 and higher) for load testing of Ajax applications.

What is Ajax?

[From Wikipedia, the free encyclopedia](#)

Ajax, shorthand for *Asynchronous JavaScript and XML*, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is meant to increase the web page's interactivity, speed, and usability.

The Ajax technique uses a combination of:

- XHTML (or HTML) and CSS, for marking up and styling information.
- The DOM accessed with a client-side scripting language, especially ECMAScript implementations such as JavaScript and JScript, to dynamically display and interact with the information presented.
- The XMLHttpRequest object is used to exchange data asynchronously with the web server. In some Ajax frameworks and in certain situations, an IFrame object is used instead of the XMLHttpRequest object to exchange data with the web server, and in other implementations, dynamically added <script> tags may be used.
- XML is sometimes used as the format for transferring data between the server and client, although any format will work, including preformatted HTML, plain text, JSON and even EBML. These files may be created dynamically by some form of server-side scripting.

Like DHTML, LAMP and SPA, Ajax is not a technology in itself, but a term that refers to the use of a group of technologies.

JSON

JavaScript Object Notation (JSON) is a lightweight format for representing objects and their state. Major technology providers, such as Yahoo WS and Microsoft ASP.NET, have chosen JSON for client-server data exchange as an alternative to XML, because it can be parsed more easily than XML. For example, JSON objects can be de-serialized by simply passing them to

the JavaScript **eval** function. This is a great advantage for WebLOAD users since JavaScript is our standard scripting language.

Testing Ajax applications

The overall process of testing Ajax applications is no different than testing any other web application. We start by authoring a test agenda (WebLOAD test script), debugging it in the WebLOAD IDE authoring environment, and running the test session on the WebLOAD Console. The rest of the document shall focus on how to best work with the new IDE to efficiently create an Ajax based agenda script.

Configuration of recording options

By default, WebLOAD supports most Ajax applications as long as they use text, XML or JSON data types. This means that most users **do not need** to change anything in the recording options. If you are still having trouble getting your Ajax calls recorded see the troubleshooting section later in this document.

Parsing responses from Ajax applications

Setting the SaveSource option

Ajax applications return valuable data as a response to the Ajax request. In most test scenarios we would like to parse and parameterize the response. By default, WebLOAD does not save the text of the response as is. Instead, WebLOAD uses the response to build an internal DOM object. If you want to manipulate the Ajax response returned from the server, the **SaveSource** option must be turned on. The SaveSource option instructs WebLOAD to store the complete downloaded HTML source code in an HTTP command:

- False - Do not store the source HTML. (default)
- True - Store the source HTML in document.wlSource.

When SaveSource is enabled (True), WebLOAD automatically stores the downloaded HTML whenever the Agenda calls the wlHttp.Get() or wlHttp.Post() method. WebLOAD stores the most recent download in the document.wlSource property, refreshing it when the Agenda calls wlHttp.Get() or wlHttp.Post() again. The stored code includes any scripts or other data embedded in the HTML. The Agenda can retrieve the code from document.wlSource and interpret it in any desired way.

To set the SaveSource option for a specific request, use the following code:

```
wlHttp.SaveSource = true
```

To set the SaveSource option for all requests in the agenda, use the following code:

```
wlGlobals.SaveSource = true
```

To retrieve the content of your response use the `document.wlSource`. For example, if the response contains JSON formatted data, use the following code:

```
var MyJSON = eval ( "(" + document.wlSource + ")" )
```

The same approach can be used to load your response from the `document.wlSource` to the XML parser of your choice or use the built in parser `WLXmlDocument`.

Working with different data types

WebLOAD supports most data type formats sent between the client and the server. The following table summarizes the different formats and the support for parsing and manipulating them inside the agenda script.

Format	Parsing & Access Level from script	Can be modified dynamically?
XML	XML DOM	Yes
JSon	JavaScript Object	Yes
WSDL	XML DOM	Yes
Other text-based formats	JavaScript string & regular expressions functions	Yes
Binary formats	Can be played back exactly as recorded	No

Working with Get request instead of Post request

So far we set the different settings relevant to Ajax calls implemented as Post requests. Sometimes you might want to have a Get request running as a backend Ajax call. To implement a Get request, you need to first manually set the content type, since the WebLOAD **Post Data** tab option is relevant only to Post commands (as its name implies).

```
wlGlobals.ContentType = "application/json; charset=utf-8"  
wlHttp.Get("http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/  
WebService.asmx/EchoStringAndDate")
```

Recording the session

To record an Ajax application just click on the record button and browse to your site. You should see your client/server Ajax calls recorded in the IDE. Note that any client side Ajax operation such as drag & drop or auto-complete has no effect on the server and is therefore not recorded in the performance agenda script.

Debugging an Ajax enabled agenda script

If you want to manipulate your response returned from the server you can set a breakpoint just after the `eval` operation and use the watch window to explore the content of your response (in our previous example, stored in the `MyJSON` object.)

The complete sample

We looked at one of the pages on the Microsoft official ASP.Net Ajax site and took the "Calling web methods" sample page (<http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/CallWebServiceMethods.aspx>)

```
/****** WLIDE - URL :
http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/WebService.aspx/GetServerTime - ID:2 *****/
wIGlobals.GetFrames = false
// Setting the SaveSource option
wIGlobals.SaveSource = true
wIHttp.Post("http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/WebService.aspx/Add")
MyJSON = eval ( "(" + document.wISource + ")" )
InfoMessage(MyJSON)

/****** WLIDE - URL :
http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/WebService.aspx/EchoStringAndDate?dt=%22%401167602400000%40%22&s=%22%20Happy%22 - ID:9 *****/
wIHttp.Header["Referer"] =
"http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/CallWebServiceMethods.aspx"
wIHttp.FormData["dt"] = "\"@1167602400000@"
wIHttp.FormData["s"] = "\" Happy\""
wIGlobals.ContentType = "application/json; charset=utf-8"
wIHttp.Get("http://ajax.asp.net/docs/Samples/Sys.Net.CallWebServiceMethods/cs/WebService.aspx/EchoStringAndDate")
```

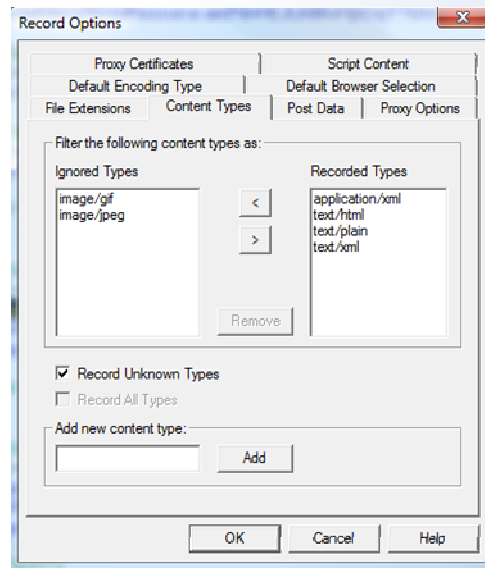
Troubleshooting

Most Ajax applications use JSON, XML or text data types between the client and the server. By default, WebLOAD supports all of them. If for some reason your Ajax request is still not getting recorded in the agenda, the recording options can be changed using the Tools - > Record Options dialog box.

Setting content type options

If you don't know exactly what content type your application uses, the best way to start recording is by turning "On" the "**Record Unknown types**" in the record options.

This setting is sufficient for most known applications. If you still don't see your Ajax calls recorded in your application you can try and check the "**Record Unknown Extensions**" in the File Extensions tab.



Future enhancements

Our continual development efforts include a number of features to further enhance the already powerful Ajax performance testing capabilities offered in WebLOAD 8.0. Our load engine technology already enables control of browser caching and multiple thread execution among other features. We are evaluating the addition of support for asynchronous calls in a future release in support of a recently introduced Ajax feature. This new Ajax method performs asynchronous calls to the server and receives callbacks once the server is done processing the request. Our current version of the engine executes the agenda in a synchronous manner and does not support asynchronous calls. In practice, this has very little effect on real performance testing of the server application since statistically the different virtual clients shall distribute evenly. Furthermore, the WebLOAD Cruise-Control feature can be used to achieve the target load. Future versions of the engine might support execution of different calls in the same agenda in parallel.

Other enhancements considered for our scripting language include the addition of support for JSON objects and XML data, providing better and easier access for manipulation of such data.

Additional resources

- 1) JSON site - <http://www.json.org/>
- 2) Microsoft official ASP.NET AJAX site <http://ajax.asp.net/Default.aspx>
- 3) Wikipedia [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))

Ajax Matters site <http://www.ajaxmatters.com/>

Contact Information:

North America	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
United Kingdom	RadView Software (UK) Email: info@RadView.com Phone: +44-080-81011165
Other Countries	RadView Software Ltd. 14 Hamelacha Street Rosh Haayin 48091, Israel Phone: +972-3-915-7060 Fax: +972-3-915-7683

RadView corporate website: www.radview.com

WebLOAD community website: www.webload.org