

# Load Testing Best Practices Using WebLOAD



# Table of Contents

WebLOAD Components and Architecture.....	4
■ WebLOAD IDE .....	5
■ WebLOAD Console .....	5
■ Load Generator (LG) .....	6
■ Probing Client .....	6
■ WebLOAD Analytics.....	6
■ WebLOAD ControlPad .....	6
■ PostgreSQL DB.....	6
■ TestTalk.....	6
WebLOAD Deployment.....	7
■ WebLOAD Console .....	7
■ Load Generator and probing client machines.....	7
■ Test Talk.....	7
■ PostgreSQL DB.....	7
■ WebLOAD ControlPad .....	8
Planning Your Tests.....	9
■ Building Testing Environment.....	9
■ Scenarios Planning.....	10
■ Defining Success Criteria.....	10
■ Uptime Planning.....	10
■ Full Test vs. Component Test .....	10
Building Agenda Using WebLOAD IDE .....	11
■ Record.....	12
■ Script Manipulation.....	13
■ Correlation .....	15
■ Parameterization.....	17
■ Response Validation.....	18
■ Testing Your Script .....	19

```
public static void falseSwap(int x, int y)
{
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
    int temp = x;
    x = y;
    y = temp;
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
}

go, x: " + x + " y: " + y);
public static void moreParameters(int a, int b)
{
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
    a = a * b;
    b = 12;
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
    falseSwap(b, a);
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
}

falseSwap, x: " + x + " y: " + y);
" b: " + b);
+
" + x + " y: " + y);
" b: " + b);

public class PrimitiveParameters
{
    public static void go()
    {
        int x = 3;
        int y = 2;
        System.out.println("in method go, x: " + x + " y: " + y);
        falseSwap(x, y);
        System.out.println("in method go, x: " + x + " y: " + y);
        moreParameters(x, y);
        System.out.println("in method go, x: " + x + " y: " + y);
    }
}

public static void go()
{
    moreParameters(a: 3, b: 12);
    int x = 3;
    int temp = x;
    x = y;
    y = temp;
    System.out.println("in method go, x: " + x + " y: " + y);
    falseSwap(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
    moreParameters(x, y);
    b = 12;
    System.out.println("in method go, x: " + x + " y: " + y);
    falseSwap(b, a);
    System.out.println("in method go, x: " + x + " y: " + y);
}
```

Executing an Agenda Using the WebLOAD Console .....	20
■ Template.....	21
■ Basic Template Definition .....	21
■ Mix.....	22
■ Template and Agenda Settings.....	23
■ Schedule Your Virtual Users & Agendas.....	24
■ Probing Client and Real User Experience.....	25
■ Cloud Load Generators .....	26
■ Performance Measurements Manager .....	27
■ APM Tools Integration .....	28
■ Test Automation and Continuous Integration.....	28
Load Test Results Analysis.....	29
■ Terminology and Meanings .....	29
■ Timers - Response Time, Hit Time, Transactions Time .....	30
■ Throughput .....	31
■ Errors of Many Kinds .....	32
■ Server Measurements .....	32
Analyze Your Test Results with WebLOAD Analytics.....	33
■ Building Templates and Portfolios.....	34
■ Publish Charts and Reports .....	34
WebLOAD ControlPad.....	35
■ Creating and editing a Dashboard .....	36
■ URL Test.....	37
■ Agenda Test .....	38
■ Template Test .....	38
■ Load Tests Management .....	38
■ Viewing Sessions .....	39
■ Sharing a Dashboard .....	39

```

public static void falseSwap(int x, int y)
{
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
    int temp = x;
    x = y;
    y = temp;
    System.out.println("in method falseSwap, x: " + x + " y: " + y);
}

go, x: " + x + " y: " + y);
public static void moreParameters(int a, int b)
{
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
    a = a * b;
    b = 12;
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
    falseSwap(b, a);
    System.out.println("in method moreParameters, a: " + a + " b: " + b);
}

falseSwap, x: " + x + " y: " + y);
" b: " + b);
+
" + x + " y: " + y);
" b: " + b);

public class PrimitiveParameters
{
    public static void go()
    {
        int x = 3;
        int y = 2;
        System.out.println("in method go, x: " + x + " y: " + y);
        falseSwap(x, y);
        System.out.println("in method go, x: " + x + " y: " + y);
        moreParameters(x, y);
        System.out.println("in method go, x: " + x + " y: " + y);
    }
}

public static void go()
{
    moreParameters(a: " + a + " b: " + b);
    int x = 3;
    int y = 2;
    System.out.println("in method go, x: " + x + " y: " + y);
    falseSwap(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
    moreParameters(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
}

public static void go()
{
    moreParameters(a: " + a + " b: " + b);
    int x = 3;
    int y = 2;
    System.out.println("in method go, x: " + x + " y: " + y);
    falseSwap(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
    moreParameters(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
}

public static void go()
{
    moreParameters(a: " + a + " b: " + b);
    int x = 3;
    int y = 2;
    System.out.println("in method go, x: " + x + " y: " + y);
    falseSwap(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
    moreParameters(x, y);
    System.out.println("in method go, x: " + x + " y: " + y);
}

```



# Introduction

This guide will help you plan, design and execute efficient load testing using **WebLOAD**.

It will help you:

**Understand the methodology and essential steps for conducting load testing.**



**Plan and implement your load testing efforts with WebLOAD in the most efficient way.**



If you are totally new to load testing I suggest that you read the 2 following resources below:

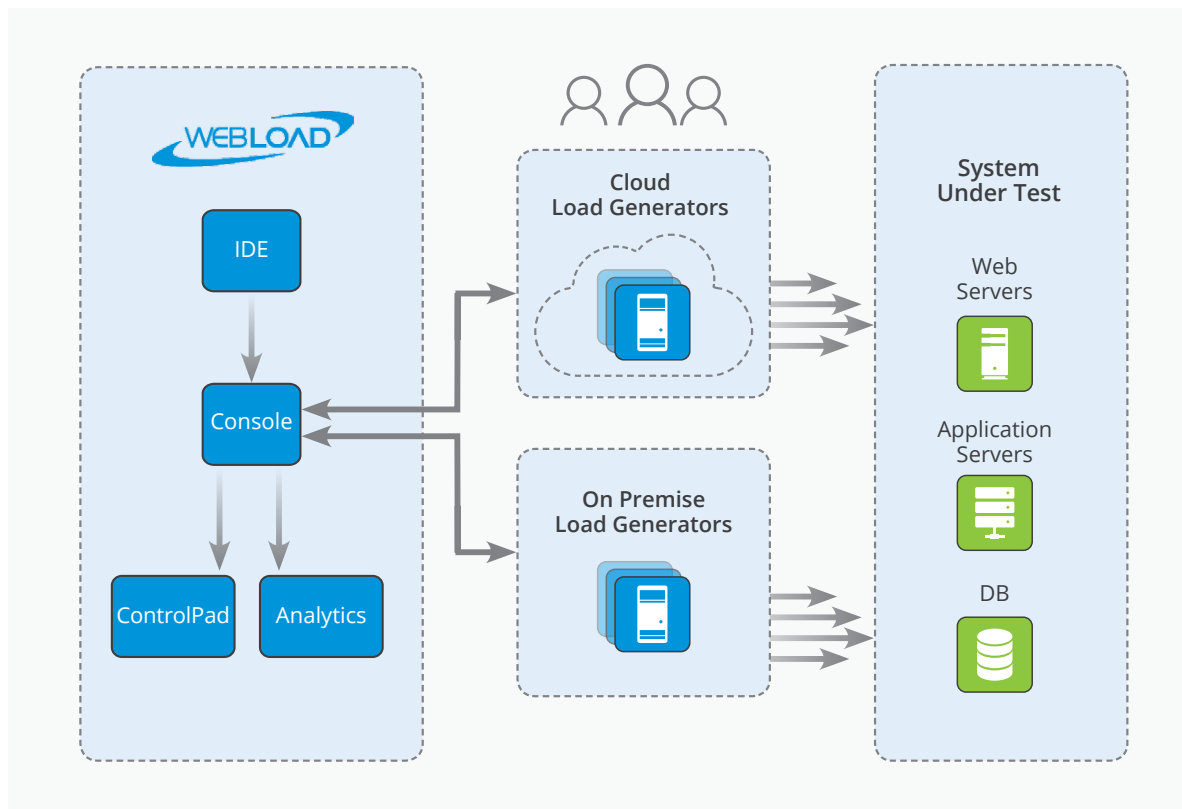
>> [What is load testing? A primer.](#)

>> [4 types of load testing and when each should be used.](#)

# WebLOAD Components and Architecture

WebLOAD's key components include: 

- **Integrated Development Environment (IDE)**  
- for developing load scenarios.
- **Console** - for managing and controlling tests.
- **Load Generator** - for generating load on your system.
- **Analytics** - for analyzing test results.
- **ControlPad** - the WebLOAD ControlPad provides a single unified command and control interface where you can create, execute, schedule, and analyze tests – all directly from your web browser.



## ■ WebLOAD IDE

You create your scenario/script - called **Agenda** in WebLOAD - via the WebLOAD IDE recorder, or directly script it using WebLOAD IDE editor. Using the IDE, you can define correlations, validations and parameterization, and define special parameters that affect the behavior of your scenarios. If these concepts are new to you, don't worry, they are all clarified later on in this document.

When you save your Agenda it is saved as a WebLOAD Project File (**.wlp**). It is recommended to test your created Agenda in WebLOAD IDE prior to the real load test. If you save your Agenda after you run it in the WebLOAD IDE it will be saved in WebLOAD Session file (**.wls**). While the .wlp file contains the agenda including all the recording information, the .wls includes all the recording and execution information that is gathered by WebLOAD IDE during the execution of the Agenda.

## ■ WebLOAD Console

Using the WebLOAD Console you define the test to be executed by defining a **Template (.tpl)**. A template, which represents a whole test, defines:

- The agendas to be executed.
- The number of virtual users to be used on a specific schedule.
- The **Load Generators (LG)** machines and cloud LG machines.
- **Probing Client** machines.
- **Performance Measurements** of your servers to be collected during your test.

Once the test execution is completed, you can save the test results into a **load session** file (**.ls**).

## ■ Load Generator (LG)

The load generators are machines that create the load on your system. Each load generator can run thousands of virtual users, and WebLOAD lets you define many load generators per each test, so you can easily reach hundreds of thousands of virtual users in a single test.

**TIP** 💡 The console itself also contains a load generator, but it is recommended to have separate load generator machines for large scale load tests. You can use the LG on WebLOAD Console for small scale tests and for validating the tests before large scale tests.

## ■ Probing Client

The probing client is a concept of running a single virtual client on a single and dedicated machine throughout the load session. The idea is to see how the load affects a standalone virtual client that is not related to the load overhead.

## ■ WebLOAD Analytics

While the WebLOAD Console lets you see test results during and after the test execution, WebLOAD Analytics lets you analyze test results in depth and create reports. WebLOAD Analytics is a comprehensive reporting framework offering many predefined and configurable charts and reports, which can be emailed or printed.

## ■ WebLOAD ControlPad

The WebLOAD ControlPad is a lightweight web server that lets you create, execute, schedule and analyze tests. You can view data from a browser or a mobile device, share results/views with your peers, and create your own views of the data.

## ■ PostgreSQL DB

WebLOAD session results are saved in files (.ls). These files are used by the console to view test results. WebLOAD Analytics and WebLOAD ControlPad use the PostgreSQL DB to view test results. After or during the test, test results are pushed to the DB, so that they can be viewed by WebLOAD Analytics and the WebLOAD ControlPad.

## ■ TestTalk

TestTalk is an internal component to WebLOAD which is responsible for the internal communication between all WebLOAD components. It must run on any machine that runs a WebLOAD component.

# WebLOAD Deployment

A typical WebLOAD deployment consists of the following:

## ■ WebLOAD Console

One or more machines that can coordinate a test, which should be installed on a Windows machine. WebLOAD IDE, LG, and WebLOAD Analytics are part of the console installation.

## ■ Load Generator and probing client machines

The number of Load Generators is not limited, and will depend on your needs. The load generator machine can be installed on a Windows or Linux machine, on premise or on the cloud.

**TIP** 💡 There are also pre-installed Load Generators on AWS cloud. If you have an Amazon account, WebLOAD can start LG machine on Amazon automatically for you during test initiation. It also stops the machine when the test ends.

## ■ Test Talk

TestTalk is WebLOAD's internal communication component. It is installed with all other components and uses ports 9000 and 9010 by default. Make sure that no other application is using these ports, or that the firewall is not blocking these ports. TestTalk ports can be changed via configuration file (WebLOAD.ini).

**TIP** 💡 It is recommended to run TestTalk as a service on

Windows or as a daemon on Linux. You can apply the option during installation. Note that TestTalk service requires administrator rights.

## ■ PostgreSQL DB

WebLOAD Installs the PostgreSQL DB as part of the WebLOAD Console installation. This provides the ability to work on the same machine with WebLOAD Console, WebLOAD Analytics and WebLOAD ControlPad.

You can also work in a distributed mode, where you install several WebLOAD consoles, WebLOAD Analytics and the WebLOAD ControlPad on different machines. In this case you may prefer to have a centralized PostgreSQL DB installation, with all WebLOAD components connected to it. This can be configured via the following settings:

WebLOAD Console - Tools > Global Options > Database

WebLOAD Analytics - Window > Preferences > Database

WebLOAD ControlPad - Attaching to the PostgreSQL DB is done via dashboard.bat



## ■ WebLOAD ControlPad

The WebLOAD ControlPad is typically installed as part of the console machine but it can also be deployed on a dedicated machine. The WebLOAD ControlPad is automatically attached to the local PostgreSQL.

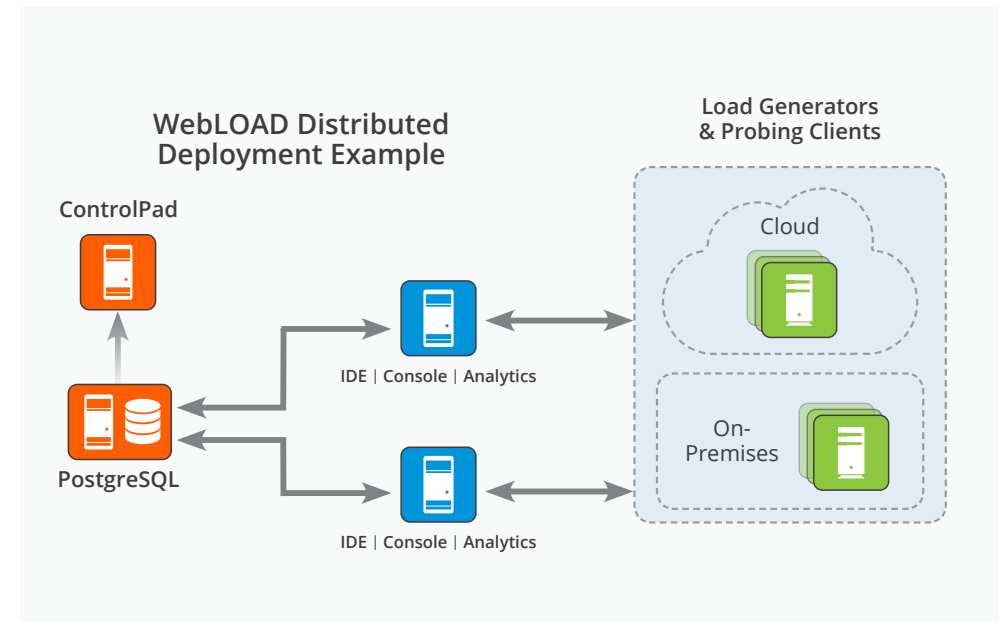
In order to show currently running sessions - In WebLOAD Console, go to

Tools > Global options > Database

and set the checkbox 'Insert Statistics into PostgreSQL database during the session'.

The screenshot shows the 'Global Options' dialog box with the 'Database' tab selected. The 'Insert into database' section has two checked checkboxes: 'Insert statistics into Postgres database during the session' and 'Insert PMM data into database'. The 'Database configuration' section contains the following fields: Database host name: localhost, Database port: 5432, Database name: radview, User name: reporter, and User password: (masked with dots). A 'Test configuration' button is located below the password field. At the bottom of the dialog are 'OK', 'Cancel', and 'Help' buttons.

The following diagram illustrates example of distributed deployment of WebLOAD:



# Planning Your Tests

Before you get into your car, you always know where you're going. The same applies in load testing. Planning is critical and prior to creating any test, you should ask yourself:

What do you need to run?

What is the load that you want to simulate?

For how long?

What are the measurements of success or failure?

What are your expected results?

All of these and more must be clear ahead of time. Otherwise, you'll find yourself running around in circles without making any real progress.

**A full blown book can be written on planning load testing but here are a few essential guidelines:**

## ■ Building Testing Environment

Build a testing environment that simulates your real life environment. For more information, see this blog post: <http://www.radview.com/6-tips-for-building-a-better-load-testing-environment/>.

**TIP** 💡 Do not forget to simulate any periodical/background processes in your real environment. They may affect your users while they are running.

**TIP** 💡 You may need to disable some features in your application. For example, if you have integration with a payment system, you do not want your test to make a payment. Or if you simulate book ordering, you do not want items to be shipped to virtual users!

## ■ Scenarios Planning

Plan for your users' scenarios. Make sure you cover the following questions:

- A. How many users are working in parallel?
- B. Which scenarios are they running? What is the distribution of the scenarios between the users?
- C. From which locations? What is the distribution of the scenarios between locations?
- D. Which device are your users using? In which distribution?
- E. What is the expected load during peak times? For example, a certain day like Black Friday. Another example may be stock buy/sell requests that may be higher at the beginning of day, or week. In other words, when you plan your test schedule, it should be based on the behavior of your system in real life.

For more information about planning and creating scenarios see: <http://www.radview.com/how-to-plan-realistic-load-test-scenarios/>.

## ■ Defining Success Criteria

Define success criteria that will be based on:

- A. Successful transaction percentage
- B. Expected response time
- C. Expected response validation errors percentage

You may have different success percentages for different transactions from different locations.

## ■ Uptime Planning

Test duration. How much uninterrupted uptime does your system require? This affects how long your test should run. For example, if you have an online store that needs to work 24/7, it may be smart to test it for a few weeks to ensure response time or memory usage does not deteriorate during long runs.

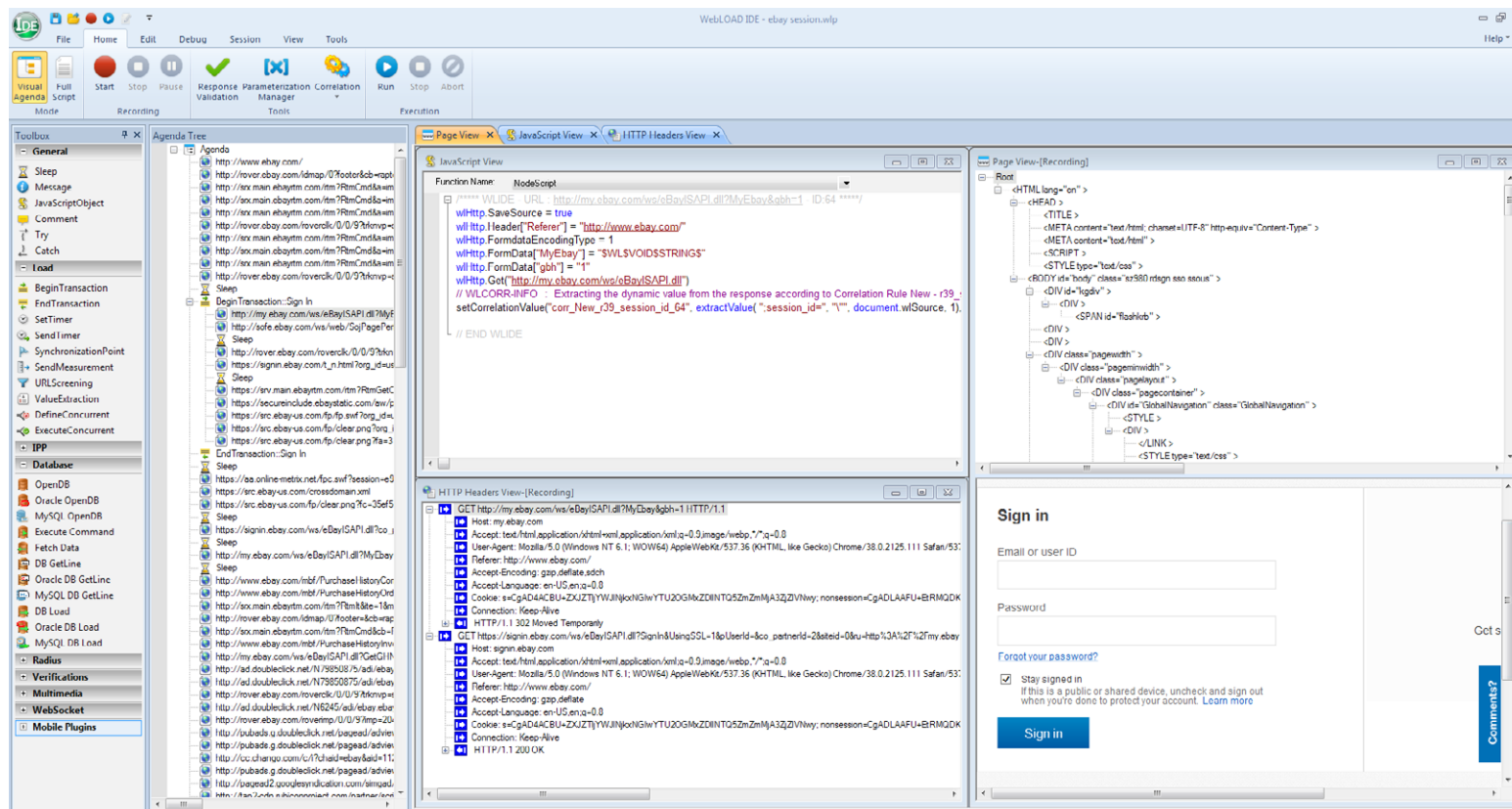
## ■ Full Test vs. Component Test

You may decide to test your system in several ways:

- A. Testing the entire system from the cloud - this will ensure that you are addressing all layers: the internet, load balancer, web servers, web apps, etc.
- B. Testing the system from your internal network, skipping the internet. You can even skip the load balancer and test your system starting from your web server.
- C. Test specific layers inside your system, for example web services or even a specific DB query.

# Building Agenda Using WebLOAD IDE

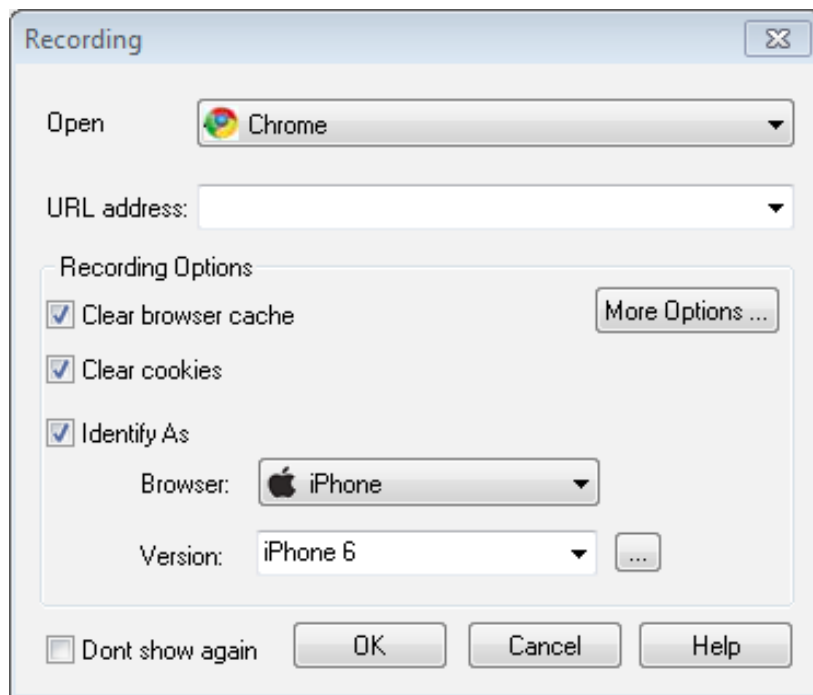
Use the WebLOAD IDE to create Agendas to be later executed by the WebLOAD Console.



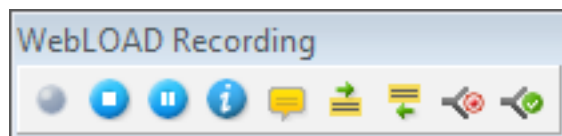


## ■ Record

The most common usage is to record your agendas using WebLOAD proxy recorder. When you click the record button, you can select which browser to use for the recording, and you can also identify your session as it was done by a browser or by a mobile device.



Make sure that a small toolbar named WebLOAD Recording is opened automatically where you can control the recording, add “transactions”, add “messages”, etc.




WebLOAD records your activity in “Incognito mode” in Chrome (“In-Private” in IE), which means that your cache and cookies are ignored. In this situation your activity is recorded as a new user who is doing an activity.

**TIP** 💡 HTTP traffic is being recorded by WebLOAD’s proxy, using the default port 9884. If this port is used by another process, change it via

Tools > Recording and Script Generation Options > Proxy Options



## Tips

- **Begin/End Transaction.** Transaction is an important term in load testing, as it represents a certain activity that you might want to measure, for example, “login”, “buy stock”, etc. That activity can be built from one HTTP request, or a few HTTP requests or others. You might want to wrap the relevant HTTP requests with Begin/End Transaction. 
- **Filtering.** The proxy recorder records all the traffic that goes from the browser. There are redundant URLs that you may want to filter out from your test. For example, there are ads that are directed to an external site, or a browser plug-in that performs some activity. WebLOAD can filter these activities from the recorded session. Go to

Tools > Recording and Script Generation Options > URL Filtering

## ■ Script Manipulation


### Agenda Tree Pane and JavaScript View

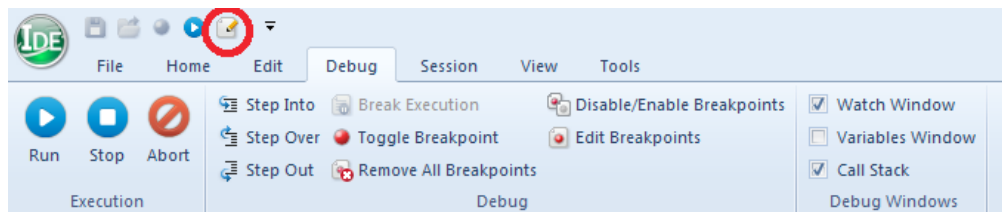
You can see the recorded agendas via 2 main views:


#### ■ Tree Pane on the left


#### ■ JavaScript View on the right

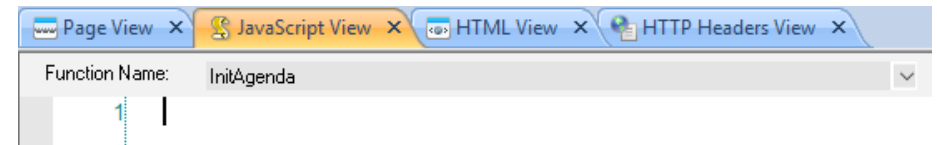
These 2 views are correlated. Clicking on a specific node on the tree will present the node related script on the JavaScript View to be edited. Clicking on the tree root will present the entire JavaScript code. To edit the entire script, press the “full script” button.

**TIP**  WebLOAD IDE has two working modes: edit mode and execution mode. When you are in execution mode, you cannot make any change to your Agenda as it affects the execution. So, before executing, WebLOAD asks you to save all your work and it moves you automatically to execution mode. If you want to edit your Agenda, switch back to edit mode.



**TIP**  Right click in your script to easily insert WebLOAD variables and methods to your script.

**TIP**  **for advanced users:** WebLOAD has special methods such as InitAgenda that runs once for each execution, or InitClient that runs once per each client, etc. You can create these methods by selecting these Functions at the top of the JavaScript View, and write your JavaScript special code. To learn more about WebLOAD methods flow, it is recommended to drill down into the documentation.



### Toolbox Pane

To manipulate the JavaScript code, you can also use the toolbox on the left panel. Just drag and drop a certain building block to the tree. Once you're done, new node will be added to the tree. The most useful building blocks are the Begin/End Transaction and Set/Send timer. All building blocks are also available in direct scripting.

## Advanced Scripting - JavaScript, DOM, XML, & Java



WebLOAD uses the Standard JavaScript Language with many dedicated objects and methods that help create tests - starting with Objects such as wlHttp, wlFTP and other protocols, wlGlobals for setting global variables, special methods for Messages such as InfoMessage and ErrorMessage, Transaction Methods, etc.

WebLOAD also creates DOM for each Http response, which provides many capabilities to manipulate your response by scripting special verifications, etc.

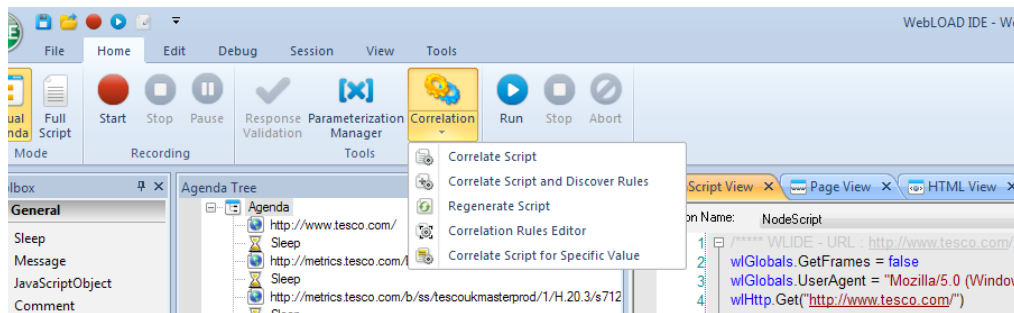
You also have direct access from JavaScript to Java, COM, and XML Objects. This lets you call external code, and enhance your script with non-standard APIs and protocols.

### Tips

- It is recommended to go over the WebLOAD Scripting Guide doc, in order to learn more about these scripting methods and many more.
- In order to code in your script, you can add a code to any recorded node, but it may be convenient also to drag and drop JavaScriptObject to the tree, and you can code there anything you'd like. This way you can keep the structure of the tree while adding the JavaScript segment you need.
- You can save your reused JavaScript methods in a .js file, and when needed use the IncludeFile method. A good example for such a usage may be login and logout which are common to all application activities.

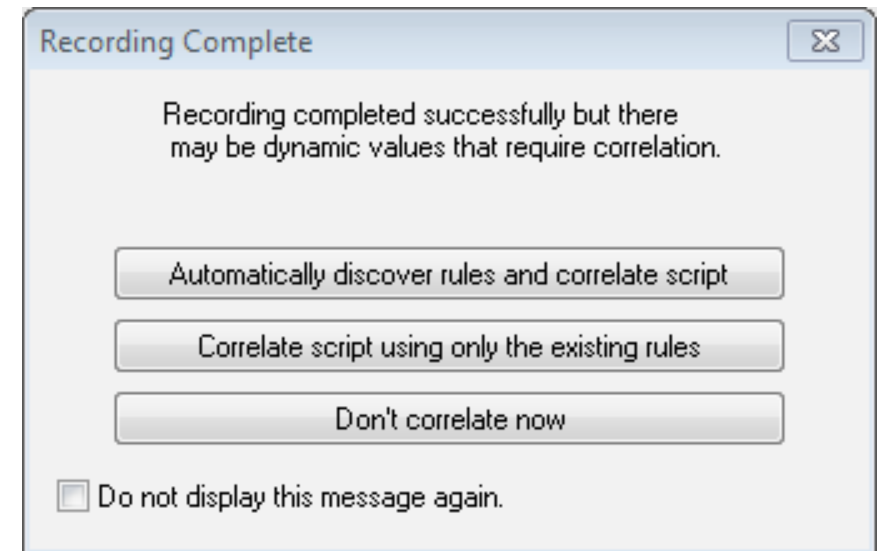
## ■ Correlation

Correlation is an important concept in Load Testing. It addresses the 'conflict' between the fact that servers keep track of client sessions and add dynamic values to the client's sessions on one hand, and the fact that these dynamic values become static in a recorded script on the other hand. The script needs to be changed in order to get rid of these values and make the script dynamic values 'friendly'. A good example of this is SessionID, which is unique to every client. It cannot use the recorded SessionID. Doing so will cause errors.



WebLOAD's correlation engine helps you find the 'problematic' values within the recorded script and replaces them automatically. The correlation uses a rule based engine that has default rules, and can be enhanced with any new rule that you find necessary. Once the rules are set correctly, you do not need to deal with the dynamic values ever, since after WebLOAD learns your application, the entire process is 100% automatic.

Once the agenda recording is done, a correlation window will be opened. If it is the first time you have recorded your application, click on the button "**Automatically discover rules and correlate script**".





# Building Agenda Using WebLOAD IDE

This will open a screen with all fields which are suspected as dynamic values that need to be correlated. Now, you can mark which rule should be used.

Correlation Engine Results

Correlation process completed. Please select the appropriate dynamic values to correlate from the list below.

Auto-discovery found 16 new rules, added to group 'New1'. Note that some rule types are not automatically discovered.

See [the online help](#) for more information on correlation in WebLOAD.

	Use	Field Name	Value	Node ID	Node URL	Rule Group	Rule Name	Add as permanent
1	<input checked="" type="checkbox"/>		a4573498f791b8522de53649a15d.lb2	2	GET http://198.11.247.44/va	Java	JSESSIONID_cookie	Active
2	<input checked="" type="checkbox"/>	ORA_ADF_V	4332958613122431	2	GET http://198.11.247.44/va	New1	r0_ORA_ADF_VIEW_LOOPB	Temporary
3	<input type="checkbox"/>	javafx.faces.	l-j7vrbndle	3	GET http://198.11.247.44/va	New1	r1_javafxfacesViewState	Temporary
4	<input type="checkbox"/>		type	3	GET http://198.11.247.44/va	New1	r2_eventb1	Temporary
5	<input checked="" type="checkbox"/>	ORA_ADF_V	4332994782028048	7	POST http://198.11.247.44/	New1	r3_ORA_ADF_VIEW_REDIRE	Temporary
6	<input type="checkbox"/>	javafx.faces.	l-j7vrbndld	9	GET http://198.11.247.44/va	New1	r1_javafxfacesViewState	Temporary
7	<input type="checkbox"/>		type	9	GET http://198.11.247.44/va	New1	r2_eventb1	Temporary
8	<input type="checkbox"/>		immediate	9	GET http://198.11.247.44/va	New1	r4_eventd1	Temporary
9	<input type="checkbox"/>	oracle.adf.vi	pt.mr:0:r1	9	GET http://198.11.247.44/va	New1	r5_oracleadfviewrichRENDER	Temporary
10	<input type="checkbox"/>	event	pt.mr:0:r1:0:13	9	GET http://198.11.247.44/va	New1	r6_event	Temporary
11	<input type="checkbox"/>	event	longRunningPopup	9	GET http://198.11.247.44/va	New1	r7_event	Temporary
12	<input type="checkbox"/>	oracle.adf.vi	longRunningPopup	9	GET http://198.11.247.44/va	New1	r8_oracleadfviewrichPROCE	Temporary
13	<input type="checkbox"/>	event	pt.mr:0:i6	9	GET http://198.11.247.44/va	New1	r9_event	Temporary
14	<input type="checkbox"/>	event	pt.mr:0:i10	9	GET http://198.11.247.44/va	New1	r10_event	Temporary
15	<input type="checkbox"/>	event	pt.Menu1:1:item1	9	GET http://198.11.247.44/va	New1	r11_event	Temporary
16	<input type="checkbox"/>	event	pt.cmi1	9	GET http://198.11.247.44/va	New1	r12_event	Temporary
17	<input type="checkbox"/>		immediate	13	POST http://198.11.247.44/	New1	r4_eventd1	Temporary
18	<input checked="" type="checkbox"/>	ORA_ADF_V	4333039170786032	18	POST http://198.11.247.44/	New1	r13_ORA_ADF_VIEW_REDIRE	Temporary
19	<input type="checkbox"/>	javafx.faces.	l-j7vrbndlc	20	GET http://198.11.247.44/va	New1	r1_javafxfacesViewState	Temporary
20	<input type="checkbox"/>	oracle.adf.vi	pt.mr:0:r1:0:1_id-1798455283_47e3d71f:	20	GET http://198.11.247.44/va	New1	r14_oracleadfviewrichSTREA	Temporary
21	<input type="checkbox"/>	event	pt.mr:0:r1:0:1_id-1798455283_47e3d71f:	22	GET http://198.11.247.44/va	New1	r15_event	Temporary
22	<input type="checkbox"/>	oracle.adf.vi	pt.mr:0:r1:0:12	25	POST http://198.11.247.44/	New1	r14_oracleadfviewrichSTREA	Temporary
23	<input type="checkbox"/>	javafx.faces.	l-j7vrbndlb	39	POST http://198.11.247.44/	New1	r1_javafxfacesViewState	Temporary

Rule details

Rule type: All body text

Field name: ORA\_ADF\_VIEW\_REDIRE

Original value: 4333039170786032

Prefix: >ORA\_ADF\_VIEW\_REDIRECT=

Suffix: ;

Description: A\_ADF\_VIEW\_REDIRECT=...4333039170786032...; expires=\$exp\$; pat

Use All Use None OK Cancel Help

**TIP** 💡 WebLOAD helps you define the necessary rules for future recordings. It shows for each value whether it was found by a predefined and **Active** rule. WebLOAD also shows whether a value was found by its enhanced automatic algorithm that scans the pages and looks for more sophisticated values - these values are presented as **Temporary**. If you find a temporary rule as one that should work for future recordings as well, make it **Permanent** via the dropdown menu.

**TIP** 💡 Many times, servers have a time related dynamic value. If you execute the script right after the recording (without correlating), it might execute correctly, but after an hour when the server changes the dynamic value, your script will fail. It is a good practice to test your script after correlation, to make sure the value replacements are done correctly. It is even better to test your script again a few hours later, or even a day after it was recorded and edited, to make sure it still works correctly.

## ■ Parameterization

Parameters are your way to create real life scenarios. For example, you would like to create a load test with 1000 different virtual users at the same time. You'll need to create different login details - username and password - for every virtual user. Not using parameters may cause errors in the best case or an inaccurate test in the worst case.

The Parameterization Manager on the home tab menu helps you manage your parameters. You can get parameters from a predefined file, or create dynamically dates, times, strings, etc.

Once your parameter is configured, it will be automatically added to a WebLOAD variables list and you are able to use it in your script view via

“right click” ➤ Insert Variable

similar to the way you add any other WebLOAD variable to your script. You can always code it directly as well.

Parameterization Manager

Definition

Name: Users

Type: File

Description:

Select input file

File name: C:\Customers\RadView\Client Issues\AA demo 10.3\...

File delimiter: ,

Create New Data File

Select access method

☒ Use values from the file

☐ Use values, ensures that virtual clients do not use the same value at the same time

☐ Use all values once and stop the virtual clients

☐ Custom (Advanced)

Scope: Global Order: Not ordered When out of values: Cycle values

Update policy

☒ Update value on each Round

☐ Update value on each use

☐ Update value per Virtual Client

☐ Show all file rows ☒ Show first 10 rows only

☒ Use first row as title row - When selected, the first row of the file will be used as the title row of the grid. This means that the values of the first row will not be used. Instead, these values will represent the variable name of that field.

	Users_UserName	Users_Password	Users_FirstName	Users_LastName
1	jim	1234	Jim	Brennan
2	sarah	1234	Sarah	Smith
3	sally	1234	Sally	Bronsen

Add Delete OK Cancel Help

## ■ Response Validation

In load testing it is important to validate the correctness of the responses and not just keep track of performance. This ensures that the server is functioning correctly, even under load. For more details, read the blogpost:

[www.radview.com/blog/validation-during-load-testing-trust-but-verify/](http://www.radview.com/blog/validation-during-load-testing-trust-but-verify/)

WebLOAD lets you verify response values by coding validations directly in the script or by using the Response Validation feature. When you are focused on a certain node, working in Page View or HTML View of each node, you can right click and open the response validation window. You can then define validation for a certain value in the page, set a time limit to the page, decide how to proceed when validation fails, etc.

The screenshot shows the 'Response Validation' dialog box with the following sections:

- Page Title:** Includes a 'Validate' checkbox, a text field for 'Success if Page Title is:' (containing '""'), and a label 'Recorded Page Title is: SugarCRM'.
- Page Time:** Includes a 'Validate' checkbox and a 'Page Time limit:' text field followed by 'sec'.
- Content length:** Includes a 'Validate' checkbox and three options: 'Equal to' (with a text field and 'bytes' label), 'Greater than' (with a text field and 'bytes' label), and 'Less than' (with a text field and 'bytes' label').
- Content:** Includes a 'Validate' checkbox, radio buttons for 'Success if response' (selected 'contains', unselected 'does not contain'), a text field containing '"John Demo " + MakeUnique.getValue()' with an 'Add Parameter' button, and 'Add'/'Remove' buttons. Below is a table:

Contain	Context text
Contain	"John Demo " + MakeUnique.getValue()
- In case of validation failure:** Includes three radio buttons: 'Display warning and continue running' (unselected), 'Display error and stop the round' (selected), and 'Display fatal error and stop test execution' (unselected). It also has an 'Error message (Optional):' text field containing '"failed to create new lead"' and a 'Call to JS function:' text field.

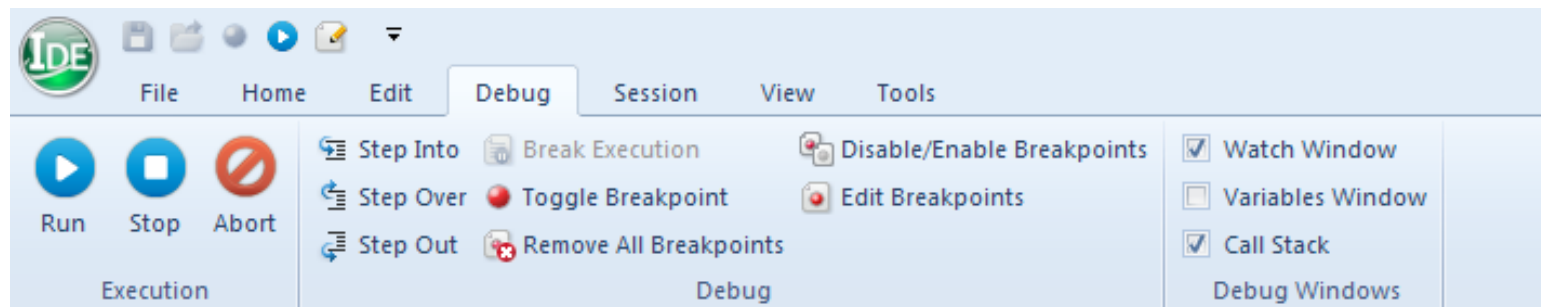
At the bottom are 'OK', 'Cancel', and 'Help' buttons.

**TIP** 💡 You can combine the response validation and the parameterization manager. As part of validating response values in your script, you can have a predefined list of known values that you expect to see in the response. For example, if you have 1000 different users for your bank application, each will have a different balance you may want to validate.

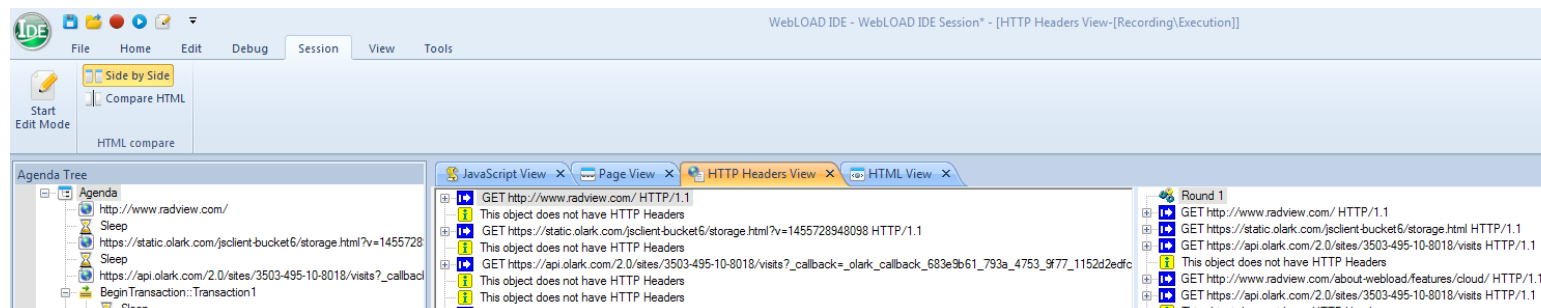
## ■ Testing Your Script

After you have made all necessary changes to your script, it is highly recommended that you will execute the script several times to make sure it works properly. You may have made many changes while developing the script, so before you load test your system with 10,000 virtual users, you want to ensure it performs correctly.

In WebLOAD IDE you can run the script from start to end directly or you can use the Debugger which lets you run the script step by step, set breakpoints, check each step, check temporary values you have set, etc.



**TIP** 💡 When you run your script you can see side by side and compare the server response during current execution and during the recording phase.





# Executing an Agenda Using the WebLOAD Console



**The WebLOAD Console lets you manage load testing execution by choosing the agendas, the load generation machines, the load profile (virtual users throughout the test), etc.**

Before defining test execution parameters, bear in mind your test goals and requirements. How many virtual users, which activities, in which percentages, from which locations, etc. In addition, server performance measurement that you would like WebLOAD to collect for you, etc. Here are a few tips:

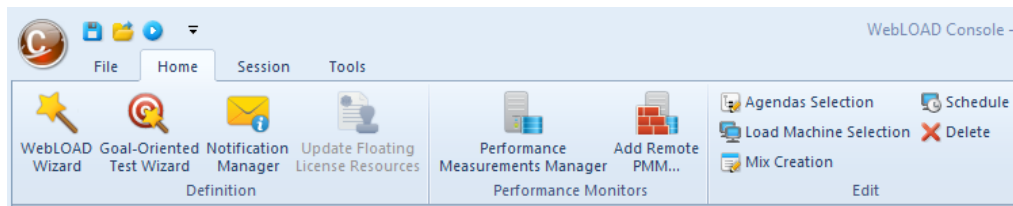
**TIP 💡** Test your system first in low volume with only one Agenda (do not mix). It is easier to find the root cause in a simple test, and in most cases there are issues that can be easily found this way. For example, it is far more difficult to find root cause when you run a mix of 20 agendas, than when running one agenda only. Start small and grow. Only after you are satisfied with the low volume, increase it and mix it to your test targets.

**TIP 💡** In the agile world you perform tests on the system while it's being developed. You don't have to wait until you have full system to start testing it. You can test performance in your first iteration, when you have one service ready, and move on from there to other services. You definitely do not need to wait until the very end of your development when performance testing at this point may be too late. With the richness and flexibility of WebLOAD scripting you can test any partial ready service and manipulate the test to your specific condition.

**TIP 💡** Document your tests. Performance testing is an iterative process and you'll be running tests multiple times, modifying settings and running again. It is easy to get confused without detailed tracking of what were the goals and exact settings each run. So document your tests - what was running, why it was running, what was changed from previous test, etc. It will save you a lot of time during the triage process.

**TIP 💡** Plan enough time for running performance tests. Running tests, making changes to the environment, fixing, re-running and analyzing results consumes time.

## ■ Template

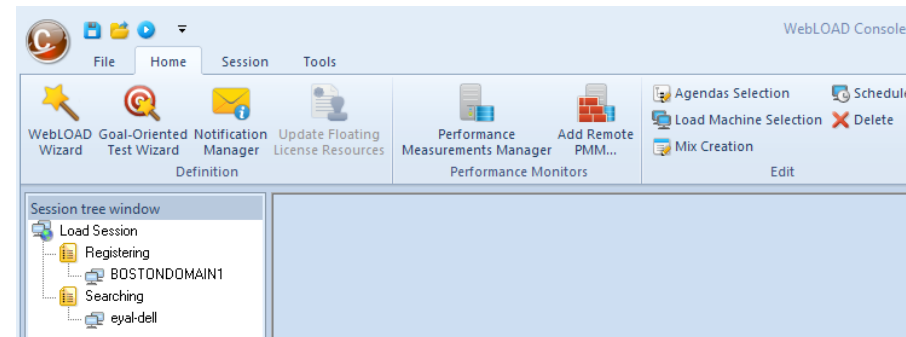


You begin execution by creating a Template in the WebLOAD Console. The template holds the configuration details of the test session: LG machines, schedule, scripts, etc. You can create a template several ways:

- Using a WebLOAD Wizard - The wizard goes through all needed options for creating a successful test session.
- Using the Goal-Oriented Test Wizard - which lets you define a test while you have your target in mind. You define the target of your test and the test will run automatically until it reaches the target. You do not need to define any scheduling - WebLOAD does that for you. For example, you would like to test how many users your system will handle while your "register transaction" is still below 2 seconds - WebLOAD will load your system gradually and will notify when it reaches your target.
- Manually creating your template.

## ■ Basic Template Definition

The template definitions are built around the following structure:



Below the load session you first add the agendas to run in the test, and for each agenda you define the Load Generator that will run it. In the example above, there are two agendas defined - Registering and Searching. Two different load generator machines will execute each agenda.

The wizards will create this structure as you progress with your definitions. If you build your template manually, you should add agendas either by right clicking the Load Session object in the tree, or by clicking the Agendas selection in the edit bar. The same way, you can add machines, schedule, change settings, etc.

## ■ Mix

In WebLOAD you can select several agendas and dedicate load generators. But you can also define a mix of agendas to run with a certain distribution between them, and easily define machines and schedule for all of them together rather than by one to one.

First, define your mix by selecting the “**Mix Creation**” button from the console top edit bar. It opens the following window:

Edit Mix

Mix name :

☒ Based on the number of virtual clients  
☐ Based on the execution time

Agenda	Weight	%	Browser	Bandwidth	Agenda Path
after1	1	50.00	As Recorded	Unlimited	C:\Users\davidb\Docume
bENNY	1	50.00	As Recorded	Unlimited	C:\Users\davidb\Docume

Buttons: Add ..., Remove, Options..., View ...

Agenda:  Weight:   %

Browser:  Bandwidth:

Version:  ... Max Connections:

Buttons: Save, Cancel, Help

Add agenda files to the mix, set their weight, and the other configurations. Once your mix is defined, save it in a .mix file. From now on, you manage the mix entity and select it in your test, like any standard agenda.

You can also add a few mix files and a few agendas in the same test. For example, executing one mix from a few machines, executing a second mix from other machines, and specific agendas on yet other machines. This powerful capability gives you the option to reach the most enhanced agenda combination you need, with the most enhanced configuration.

**TIP** 💡 Read the following blogpost to learn more about distribution of your virtual users during the test: <http://www.radview.com/how-to-distribute-stress-in-load-testing-scenarios/>.

## ■ Template and Agenda Settings

In addition to the basic template configuration, WebLOAD Console lets you fine tune settings, such as sleep time, browser cache, etc. It is possible to set options for all agendas or for specific agenda only.

Under the Tools tab, you'll find a few options:

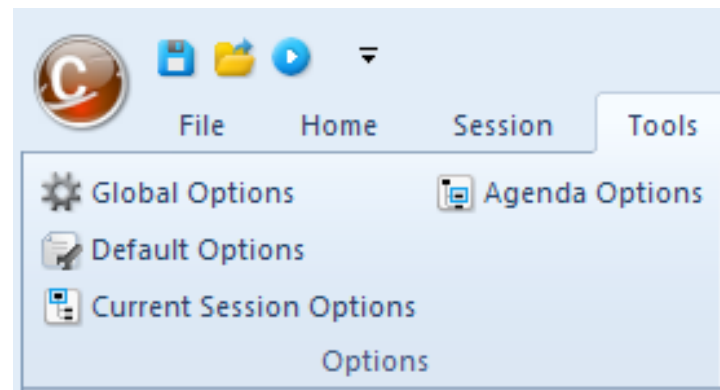
■ **Global Options** - Parameters that reflect basic configuration of WebLOAD Console, and options that affects the activity of the entire test. For example, statistics resolution, WebLOAD DB configuration, etc.

■ **Default Options** - These options are dedicated to the agenda activity itself. The settings related to the protocols, sleep time control, zip, SSL, browser cache, etc. The default settings are used for each agenda that is added to your test.

■ **Options for the current test:**

➤ **Agenda Options** - When you add a mix or agenda to WebLOAD execution tree, WebLOAD Console copies the default options to the agenda options. So to change any setting of one agenda, change the agenda options.

➤ **Current Sessions Options** - When you make a change that will be reflected on ALL agendas that are currently defined in your test, you can change it once using the current sessions options. This is a shortcut, instead of going over each agenda options and making the change one by one.



**TIP** 💡 Many options can be written in the JavaScript. If an option appears in the script, it overrides any other option that appears in WebLOAD Console.

**TIP** 💡 After you define all options and before you press the start button - check all your settings again and make sure there are no mistakes. Load tests runs for a long time, and you do not want to discover after a few hours that you misconfigured something and you wasted precious time.

**TIP** 💡 By default, WebLOAD records the sleep time (think time) between your clicks. You may want to run your test with the sleep time as recorded, or you may want to change it dynamically during execution. In the 'sleep time control' in the Agenda Options window you can manipulate the sleep to your needs, and it will overwrite the sleep time in the agenda.



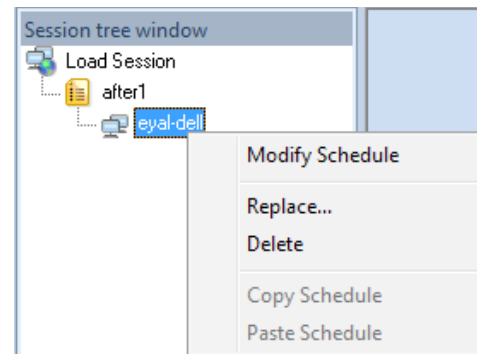
## ■ Schedule Your Virtual Users & Agendas

Scheduling can be done in two ways:

- Manual defining virtual users and times.
- Using the load profiler which contains known profiles like, steps, linear, random, etc. It helps you create complex definitions easily.

### 💡 Tips

- Do not overload your LGs. WebLOAD console lets you define as many virtual users as you want for a specific machine. Note that the machine can be “weak” and therefore cannot handle the number of virtual users you defined. If one or more of your load generators is overloaded during a test, the results may be worthless. For more information, read the following the blogpost: <http://www.radview.com/blog/how-to-to-detect-overloaded-load-generators-in-load-testing>.
- Make sure that the LG machines are “clean” from other jobs and currently running processes. This can reduce the load used power, and skew results.
- You can combine profiles in your session. For example, “step increment”, then perform “ramp up” profile, and more.



You can dynamically change the number of virtual users during test execution, overriding the template's scheduling:

Session > Throttle Control

You can change the number of virtual users.

Session > Freeze

You can freeze the progress of your session – you continue testing with the current number of virtual users and the template time does not continue.

These two features are time savers. When you want to investigate your system's behavior on specific load size, you can freeze, change the load, and continue from where you froze your test - without the need to stop the test, and execute it again.



## ■ Probing Client and Real User Experience

Probing Client is a term introduced by WebLOAD and relates to Load Generator of single client only. It is recommended to use at least one probing client in each test for the following reasons:

- Your LG machines are doing a lot of work simulating many virtual users. This means that load machines may be “noisy”. If you want to look at the response time and transaction times, or any other timers without the overhead of the load generator, it is recommended to see the results from a machine that runs only a single user. This way you can see how a single user experiences your server performance, without the “noise”.
- A machine that runs a probing client, by definition, should not be loaded by itself. Thus, the probing client machine can act as a good indication whether your servers or your LGs are overloaded. For example, you see on a certain LG machine the response time goes up while increasing the load. Does it mean that your server reached its max capacity? Not necessarily. Your LG may be overloaded (bandwidth/memory/CPU). So, take a look at the probing client. If its response time is increasing as well, it means that your server is choking. If not - your LG.

You can also put several probing client machines in different locations so you can see the results from different locations. A probing client can execute any agenda LG can execute, without any limitations.

Another interesting way to use the probing client is by executing a real browser or mobile device. This way, you load your servers with Load Generators, and measure the real user experience from a browser or from a real mobile device. WebLOAD lets you run real browsers and devices by integrating with 3rd party functional testing tools such as Selenium, Ranorex, Original Software’s TestDrive, Perfecto Mobile, etc. WebLOAD activates these tools, runs the browser activity, and collects performance statistics from the browser or the mobile device. This is also a great way to make sure that your functional tests pass/fail under load. To learn about each of the integrations, read WebLOAD documentation.

## ■ Cloud Load Generators

You can Install WebLOAD LG machines on premise and on any cloud vendor, and mix between machines in a single test.

WebLOAD also lets you run performance test via special integration with Amazon EC2. WebLOAD provides a preconfigured public Amazon Machine Image (AMI), which lets you quickly generate load with minimal setup using your own Amazon account. You add those cloud LGs during the template creation - just by adding LGs from cloud. WebLOAD automatically creates and terminates the cloud machines for your test. Your costs are based solely on your cloud machines activities during the test.

To set this up access:

WebLOAD Console > Tools > Cloud Options

The following window opens, letting you define the Amazon machines to be used.

Cloud Accounts

Account name

Saturn2

Add

Remove

Get Status

Details

Name: Saturn2

Amazon EC2 account security credentials. You can find your account information [here](#)

Access key ID:

Secret access key:

Advanced

Region: us-east-1

https://ec2.us-east-1.amazonaws.com

AMI ID: WebLOAD LG

Default

Type: M1 Extra Large

General

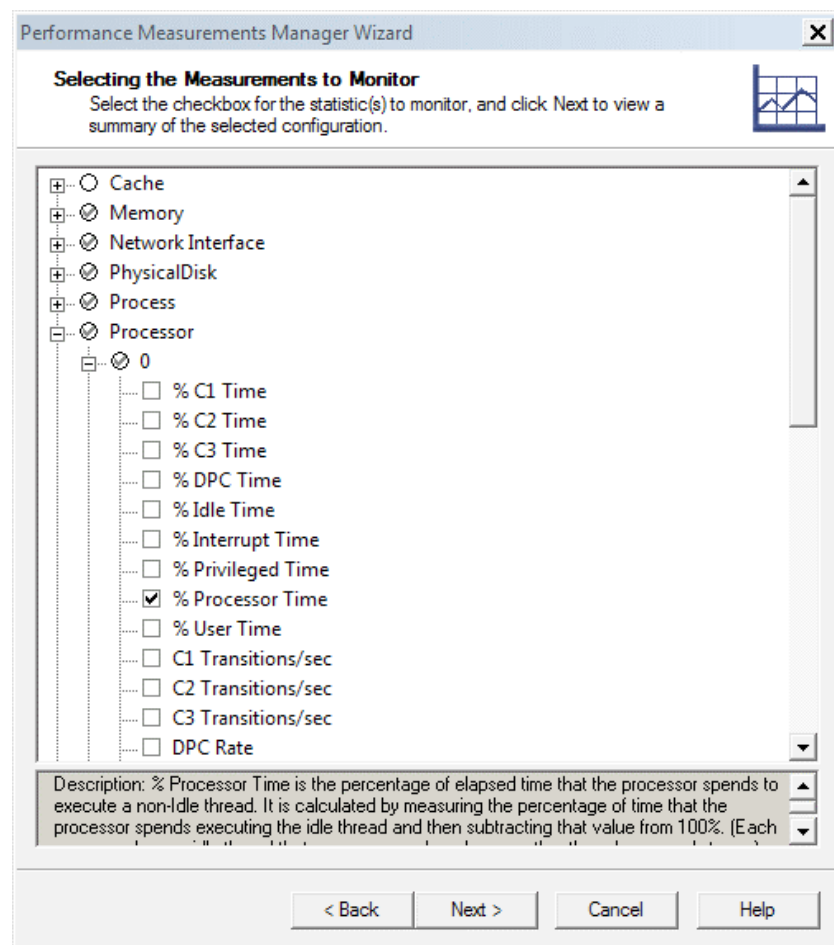
Stop cloud machines when session end: Stop Always

Apply OK Cancel Help

## ■ Performance Measurements Manager

On the WebLOAD Console home tab you can see the Performance Measurements Manager (PMM). Using the PMM, WebLOAD collects server-side performance data from operating systems, web servers, application servers, database servers, etc. This makes it possible to see client side statistics (transaction time, for example) on the same graph with server metrics and identify the root-cause of problems.

WebLOAD PMM enables direct correlation of your load scenario with data such as CPU, memory, capacity, processes, etc. You can quickly track down bottlenecks and pinpoint the weak links in your system.



Server side measurements are collected by WebLOAD Console during the test, and later are displayed as part of WebLOAD's results analytics reports. For more information on the technologies WebLOAD supports, see <http://www.radview.com/technologies-supported>.

**TIP** 💡 Do not forget to collect your server measurements during your test as it helps triage problems faster. Otherwise, you may need to run tests again just because you didn't configure the PMM or didn't collect the right information.

## ■ APM Tools Integration

WebLOAD Integrates with APM (Application Performance Management) tools such as AppDynamics, Dynatrace, and NewRelic. These tools provide deeper insight into the tested application letting you drill down to the specific line of code associated with a performance issue. To learn more see the WebLOAD documentation.

## ■ Test Automation and Continuous Integration

WebLOAD's Jenkins integration enables running any performance test scenario via a regular 'Build Step' in Jenkins.

You can define success/failures criteria based on many factors such as failure of the entire session, errors and warning, validations, performance measurements, etc. This provides the flexibility to adjust and determine how changes in performance will affect the next step in your continuous delivery pipeline.

Automated mails provide a daily summary of build performance status, comparing results to the previous automated builds and to a predefined baseline.



# Load Test Results Analysis

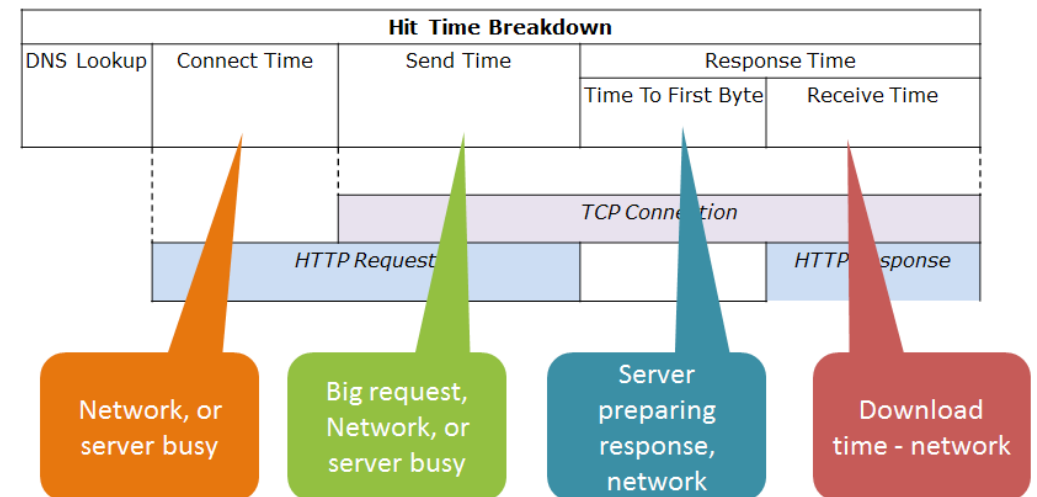
Explaining all the nuances involved in analyzing performance test results requires a dedicated book. The following, therefore, only highlights some of the key issues. For a more in-depth discussion, view our webinar [How to Interpret Load Test Results](#).

## ■ Terminology and Meanings

The diagram below illustrates the events taking place during each http request:

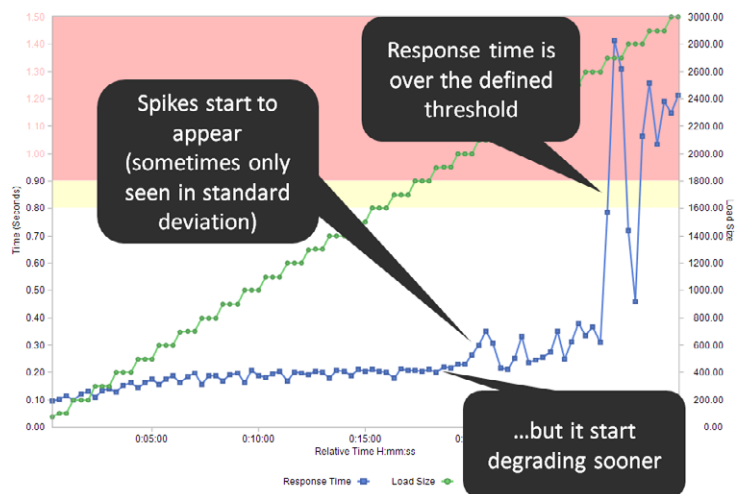
- DNS lookup, which is translated to an address via the DNS.
- A connection phase between the client and the server.
- The client sends its request to the server.
- “Time to first byte” - represents the server processing time and the network time. It means the server worked when it received the entire request, and finished when it started sending the response. This theory is true for synchronous requests.
- The complete response is sent back to the client and represents the network time.

The following diagram displays the above and suggests reasons for an issue in each step.



## ■ Timers - Response Time, Hit Time, Transactions Time

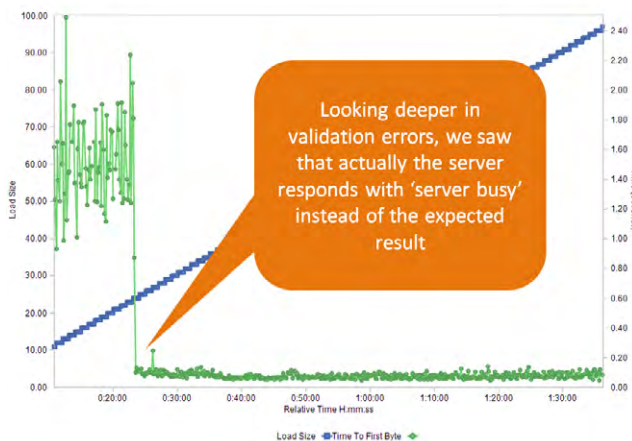
The timers are in general the first indication about your system. Of course, you should look for thresholds violations, but you may find interesting behavior even before you reached the violation.



Do you expect hit time to change when the load size is constant?



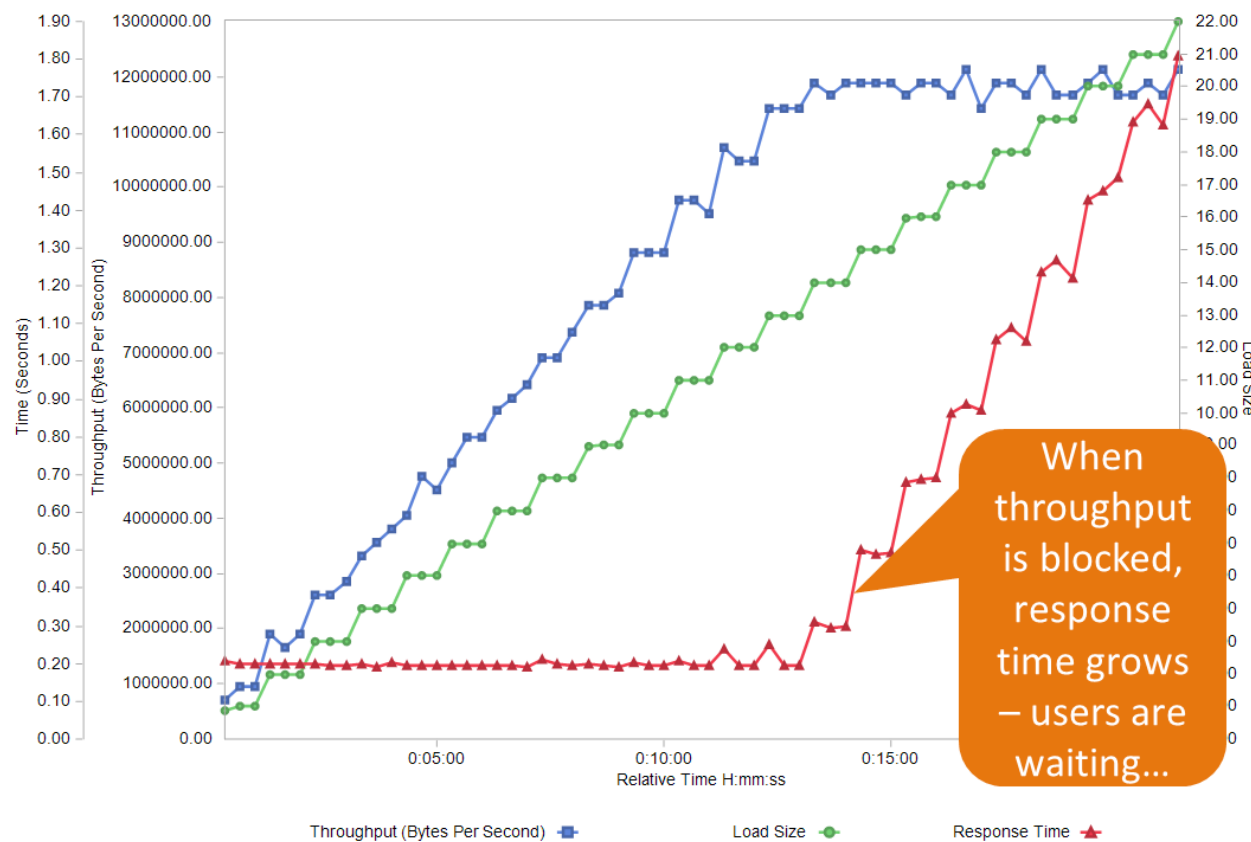
Does it make sense that the time to first byte suddenly improved significantly while the load size increased?



We can add more examples, but you can use these examples as a basic results analysis that shows where you should start looking.

## ■ Throughput

Throughput is also a measurement that tells a lot about your servers' behavior. For example, when throughput drops suddenly, or when it is steady where it should grow. Take a look at the following example how throughput is affecting response time. The system behaves perfectly at the beginning. The load size grows while the response time stays at the same level. But then, the throughput hit a limit. Once it reaches that limit, the response time grows. The reason, in this case was a network configuration issue that limited the capacity.



Another way to look at throughput is your response size expectations. For each request there is an expected response. This is important to make sure that during the load test you get the expected response. Otherwise your results may mislead you. For example, suppose you expect 1 GB as a response. You should check that you are getting the expected size. If you are getting 1MB it means that there is an issue. The Response Validation helps you easily measure the response size.

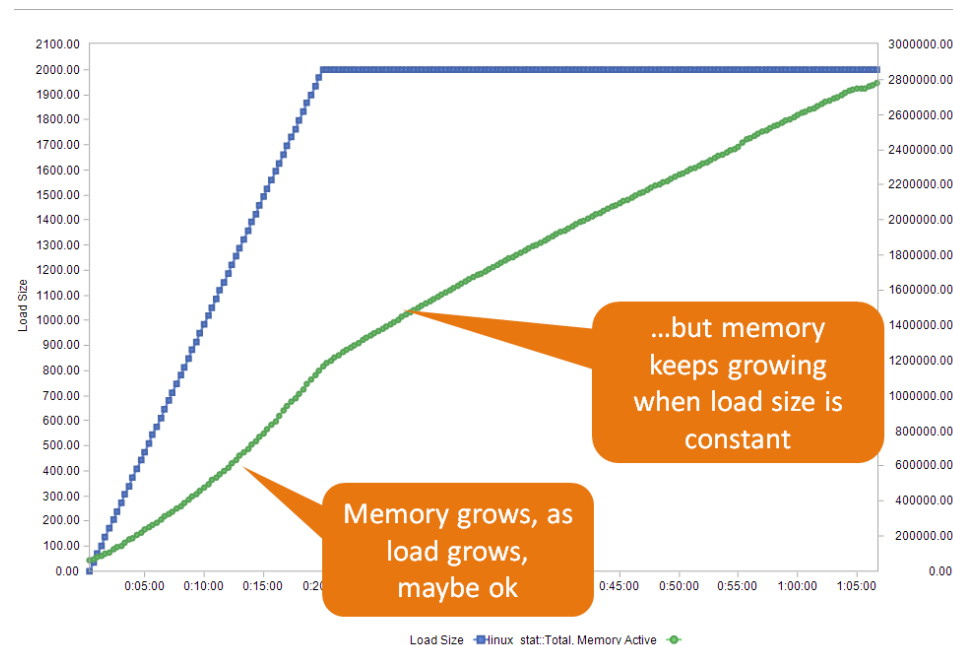
## ■ Errors of Many Kinds

We mentioned earlier in this document that you should validate the responses from the servers. This is a good habit, to check several types of errors that may occur during the test. For example:

- Server errors
- Http errors
- Validation errors - Just wrong values, or even corrupted data
- Do not forget to look at your log files. Are errors logged there?

## ■ Server Measurements

As mentioned earlier, WebLOAD helps you collect server measurements via its PMM and via integration with APM tools. These measurements with correlation with the client measurement, gives you the ability to find root cause easily. Take a look at the following example that shows how memory behaves during constant load size.

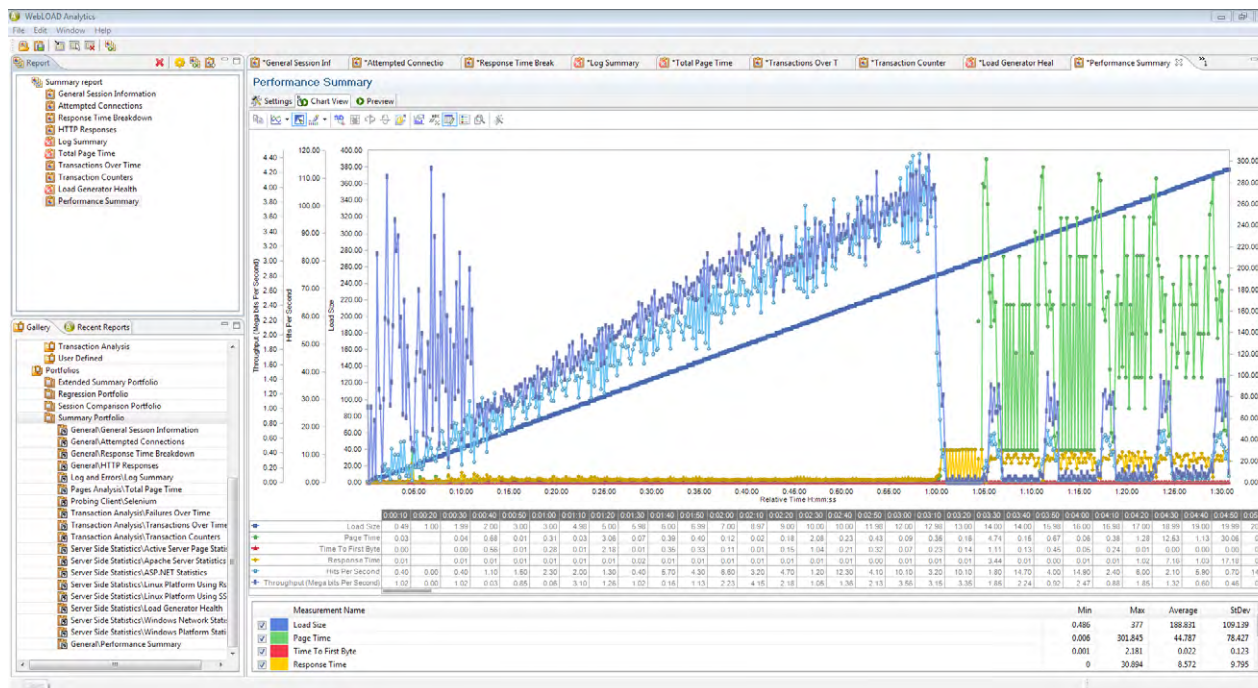


To summarize, as mentioned in the beginning of this section, there are a lot of interesting measurements and behaviors that you should look for. We tried to shed light on some important ones that will help you in your triage process.



# Analyze Your Test Results with WebLOAD Analytics

WebLOAD Analytics lets you perform deep analysis of test results once it complete running. WebLOAD Analytics provides many ready-made reports that contain many charts that can be used immediately, but you can also create your own charts and reports to suit your specific needs.



On the top left panel, you can see the currently viewed report name and its charts - in this example, "Summary Report". You can select any chart to be viewed either from the top left panel, or by selecting a tab on top of the Chart View.

You can select any report from the left-bottom "Gallery" panel. The Gallery panel is built from Templates and Portfolios. A Template is a Chart definition, while a Portfolio, which is a collection of Templates, is a Report definition. Note, that WebLOAD Analytics lets you see a full report with all its charts, or one chart only. Just double click on a Template or Portfolio in the Gallery Panel.



# Analyze Your Test Results with WebLOAD Analytics

## ■ Building Templates and Portfolios

Use the “Blank Template” to build a new one. Double click and edit it via the “settings” tab which is left of the chart view tab. Via the settings you can add measurements, zoom, change axis, etc. While you edit the Template, you create a Chart on the fly. The new Chart is added by default to the currently viewed report.

To save the Chart  
for future use select

File > Save Chart  
as Template

You can also edit one of the predefined Charts, and save the new edited Chart as a new Template in the same way

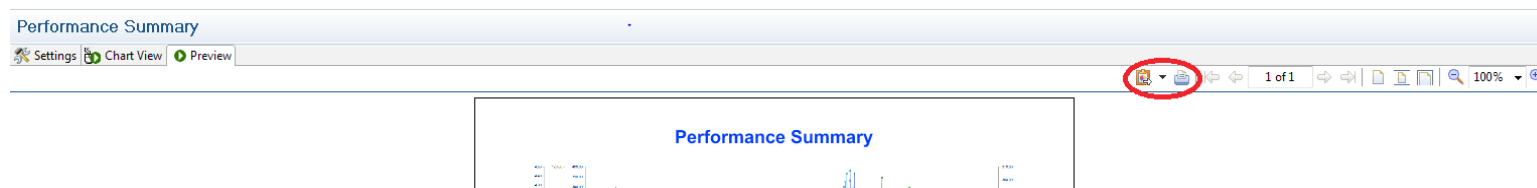
File > Save Chart  
as Template

To create new Portfolio or update a Portfolio, right click in the Gallery tree or drag and drop a Template into a Portfolio.

**TIP** ⚡ Note that all the changes you make are done on the currently viewed report, and do not affect any template or portfolio, unless you save your changes.

## ■ Publish Charts and Reports

Using WebLOAD Analytics you can create Reports to be sent to peers. Click on “Preview” button, and then you can print it or save in several formats, such as doc, pdf, html, etc.



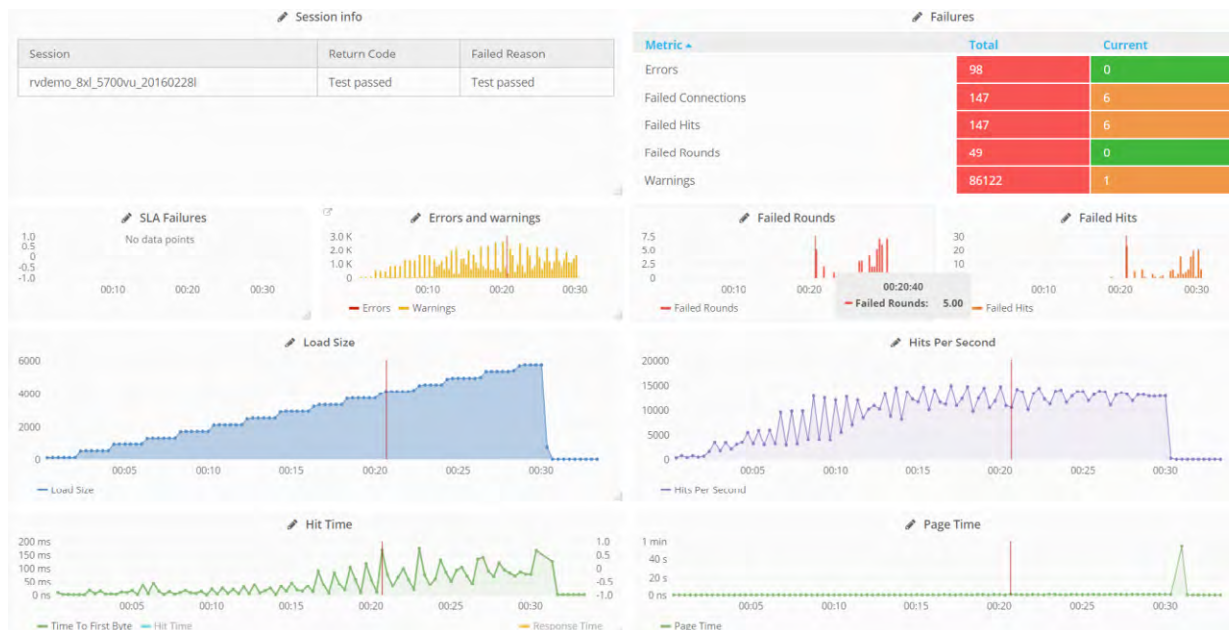
If you want to publish a full Report (and not a Chart only), use the “Publish Report” option from the File menu.

# WebLOAD ControlPad

The WebLOAD ControlPad provides a single unified command and control interface where you can create, execute, schedule, and analyze tests – all directly from your web browser, either on-premises or in the Cloud.

Some of the tasks you can accomplish using the WebLOAD ControlPad include:

- Create and edit load tests – either based on a WebLOAD agenda or by a using single URL/API
- Upload, add and manage tests, resources, and sessions
- Execute and schedule test runs
- Analyze results using ready-made and self-configured advanced dashboards
- Share tests and results with peers
- Manage access permissions for users and groups

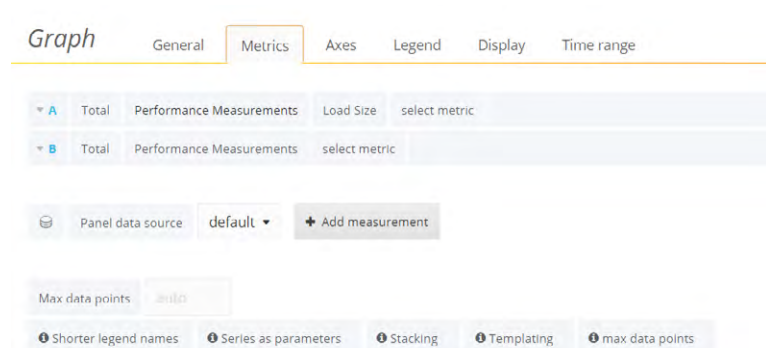


## ■ Creating and editing a Dashboard

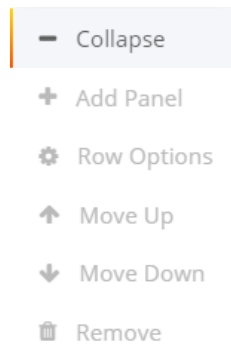
You can easily create multiple dashboards to analyze and present different aspects of your performance testing results. To edit a dashboard, click on its title and then click Edit.



You can add different measurements, and change the view to suit your needs.



You can add a new panel to a row and manage the rows, by hovering the 3 dots marker on top-left of each row.



To add a new row, select the **+ ADD ROW** button at the bottom-left of each dashboard.


Use the following icons on the top to manage your dashboards.



There are several types of tests you can create in the WebLOAD ControlPad.

## ■ URL Test

This is the most simple test you can create. Just define the the URL, virtual users, load machines, and other parameters.

 URL Load Test

Create Load Test

### General

Name

home\_page

### URL

URL

http://www.radview.com

### Load Configuration

Max Virtual Users

200

0 200 1000

Ramp up time in minutes

5

0 5 300

Time to run Max Virtul Users in minutes

60

0 60 600

Ramp down time in minutes

5

0 5 300

### Load Generators

localhost

us-east-1-aws\_1

us-east-1-aws\_2

us-east-1-aws\_3

### Recurrence

Recurrence

☐

## ■ Agenda Test

Agenda test looks like the URL test, but instead of loading a simple URL it can use a predefined Agenda that was created in the WebLOAD IDE. You will need to add your Agenda and any resources your Agenda is using to the WebLOAD ControlPad Resources section.

## ■ Template Test

Template test gives you the ability to execute a test that was created via the WebLOAD Console. You will need to add any resources your test is using to the WebLOAD ControlPad Resources section.

## ■ Load Tests Management

You can execute, analyze and manage all your tests and sessions via the the Load Tests section.

The screenshot shows the 'Load Tests' management interface. At the top, there is a search bar with the text 'website' and a '+ Add a new load test' button. Below the search bar is a table with the following columns: Name, Path, Next Execution, Last Execution, and Last Session State. The table contains five rows of test data. Each row has a set of action icons (play, stop, refresh, etc.) to the right of the 'Last Session State' column.

Name	Path	Next Execution	Last Execution	Last Session State
Website - Buy product - 10000 users	/System Tests/Buy product.tpl	Not Scheduled		
Website - Full system test - 24 hours	/System Tests/Full system test.tpl	2018-03-03 03:00	2018-02-24 03:00	Test passed
Website - Payment only - 500 users	/System Tests/Payment only.tpl	Not Scheduled		
Website - Registration - 1000 users	/System Tests/Registration.tpl	Not Scheduled		
Website - Search only - 200000 users	/System Tests/Search only.tpl	Not Scheduled		



## ■ Viewing Sessions

You can see all your sessions via the sessions view (see below) or via the Dashboards view.

*Load Sessions* Upload Session

Find by Session Name  All Statuses All Dates

Name	Test	From	To	Status	Failed Reason		Show in Dashboard
Website - Full system test - 24 hours (2018-03-18 10:28)	Website - Full system test - 24 hours	2018-03-18 10:28	2018-03-18 10:30	Test passed	Test passed		<input checked="" type="checkbox"/>
Website - Full system test - 24 hours (2018-03-10 03:01)	Website - Full system test - 24 hours	2018-03-10 03:01	2018-03-10 03:03	Test passed	Test passed		<input checked="" type="checkbox"/>
Website - Full system test - 24 hours (2018-02-24 03:00)	Website - Full system test - 24 hours	2018-02-24 03:00	2018-02-24 03:03	Test passed	Test passed		<input type="checkbox"/>
Website - Full system test - 24 hours (2018-02-18 14:15)	Website - Full system test - 24 hours	2018-02-18 14:15	2018-02-18 14:17	Test passed	Test passed		<input type="checkbox"/>
Website - Full system test - 24 hours (2018-02-10 03:01)	Website - Full system test - 24 hours	2018-02-10 03:01	2018-02-10 03:03	Test passed	Test passed		<input type="checkbox"/>
Website - Full system test - 24 hours (2018-02-03 03:00)	Website - Full system test - 24 hours	2018-02-03 03:00	2018-02-03 03:02	Test passed	Test passed		<input type="checkbox"/>

You can upload sessions that were executed by the WebLOAD Console, or view a session that is currently running in the WebLOAD Console. To do that, WebLOAD Console needs to be configured to send statistics information to the PostgreSQL DB during runtime. See the [WebLOAD Deployment section](#) for details.

## ■ Sharing a Dashboard

You can share a dashboard with your managers and peers by sending a link. You can either share a static snapshot of your current test results, or allow the results to be dynamically updated whenever the user receiving the link is running the dashboard.

Relative Time ▼ Share Zoom To Data Zoom Out 00:00:00 to 01:35:06 refreshed every 20s ▼

---

New Import Playlist

We hope this document will help you to get started with WebLOAD. If you have any questions about WebLOAD, we would love to hear from you.

Have you and your team come up with your own WebLOAD best practices?

**Share them with us!**

The WebLOAD Team



[www.radview.com](http://www.radview.com)

**North American Headquarters**

991 Highway 22 West, Suite 200

Bridgewater, NJ 08807

908-526-7756

Email: [sales@radview.com](mailto:sales@radview.com)