



CloudTest[®] Performance Testing Tutorial



CloudTest® Performance Testing Tutorial

©2012, SOASTA, Inc. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective companies.

This document is for informational purposes only. SOASTA makes no warranties, express or implied, as to the information contained within this document.

Table of Contents

Why Performance Testing?	1
Prerequisites	1
KnowledgeBase and Tutorials.....	1
Planning a Performance Test	2
Configuring Performance Monitors	3
Create a New Monitor Server and Monitor	4
Composing a Performance Test	6
Creating the Test Clip.....	6
Using the Session Template Package Wizard	14
Scanning a Clip for Values to Parameterize	15
Replace a Token using the SubString Parser.....	16
Inspecting Applied Session Template Values	20
Creating a User-Defined Validation	24
Playing and Debugging a Simple Test	25
Result Details Dashboard.....	26
Verifying Test Parameterization	28
Adding Test Complexity	32
Applying Virtual Users	32
Refining General Properties for Performance Testing	33
Refining Logging Properties for Performance Testing	34
Save Clip Element and Save Composition Details	35
Adding Additional Clips and Tracks	36
Using Locations and Servers (CloudTest Standard and CloudTest Pro Users)	37
Specifying Locations using Composition Editor (Cloud Only)	38
Using Load Composition to Prepare for Play	39
Using Play Options with Locations and Preview Mode	40
Playing the Test.....	41
Analyzing Results	42
Adding a Monitor Widget to a Dashboard	42
Debugging a Test.....	43
Tuning Performance Tests	43
Comparing Multiple Results	43

Refining Your Test.....	45
-------------------------	----

Why Performance Testing?

As soon as a load test fails, it becomes a performance test. SOASTA CloudTest supports HTTP(S) (message) and WebUI/Ajax (browser) recording, as well as SOAP/REST web service connectivity. Performance testing can involve all three of these. Performance testing is diagnostic and iterative, and answers the following questions:

- Which components of the application are the causes of a performance bottleneck?
- Can the tester isolate the components of the Web application that are the bottleneck(s)?
- Can the tester determine throughput levels for each of those components?

Prerequisites

This tutorial focuses on the analytic phases of testing. Within SOASTA CloudTest, performance testing can be performed in any mode: HTTP message recording, WebUI browser recording, as well as by other methods. This guide uses the HTTP message recordings first created in the *SOASTA CloudTest Load Test Creation Tutorial* as a starting point. Some material is repeated.

The SOASTA CloudTest Conductor is required for HTTP(S) Recording and also drives the playback of WebUI/Ajax tests and must be installed before test playback involving either type.

KnowledgeBase and Tutorials

CloudTest provides two types of recording in support of test creation. This tutorial uses an existing HTTP(S) recording. CloudTest also provides Browser Recording, which allows the user to capture user interface actions performed on a given target web site.

- Refer to the [Load Test Creation Tutorial](#) and to the CloudLink KnowledgeBase's HTTP(S) Recording section for additional HTTP(S) recording instructions.
- Refer to the [WebUI Testing Tutorial](#) and to the CloudLink KnowledgeBase's Browser Recording section for additional Browser Recording instructions.

SOASTA Conductor

The SOASTA Conductor is available on the Welcome page in the Downloads area. Refer to [Installing SOASTA Conductor](#) for detailed configuration. This application runs on all major Operating System platforms: Windows, Linux, and Mac OS X.



Performance Testing Your Web Site

Performance testing can use the exact same test clips as used within other testing types. However, within the performance context we are not as interested in a Pass/Fail result or that Pass/Fail result with respect to the load thrown at the application(s) being tested. Instead, our focus is on measuring a quantifiable result (How long does it take? and How much CPU/Memory does it take?).

A performance test differs from a load test when it comes to failures. Performance tests seek measurable data about a given application within established expectations, where they exist.

While measurable data is important in load testing, the focus is more on exploring limits. In a performance test, we want to measure how the application responds (at the CPU/memory level, at the middleware level, and so forth) at various levels of stress.

For example, a simple performance test measures what the CPU performance of the application (and environment) is when 100 and when 1000 users access a video product demo at the same time. In the following sections, we will begin with that premise, using much smaller numbers of virtual users, if any, and cover how to analyze performance against your own site and applications using SOASTA CloudTest's analytic tools.

Attacking performance problems in your web application requires a toolset capable of dealing with the complex infrastructure of a modern web application. This task is supported using the combination of recording, resource monitoring, and integrated analytics.

Planning a Performance Test

SOASTA advocates a six-step methodology for creating and executing performance tests against Web applications and services.

- Planning a Performance Test
- Configuring Performance Monitors
- Composing a Performance Test
- Running a Test
- Analyzing Results
- Tuning a Performance Test

The performance tester's task is to identify the components of a given application, if any, that are the performance bottlenecks. In the following sections, we will establish a low-end performance benchmark using 10 virtual users, and then examine the additional impact on performance as users are added. If you're using your own site, you can choose to increase these numbers, but for a public site, we'll keep the virtual users number conservative.

Once performance under stress can be evaluated, changes can be made to improve component performance and testing is repeated to determine if the changes are successful. Although your methodology will vary according to your preferences and purposes, in general the following stages are common to most performance testing:

- Define your test case benchmark

Performance testing involves positive and negative test cases that give insight into the functionality of the product being tested, usually involving some form of stress or load. Performance tests can determine the speed and resources required when the product works, or does not work.

- Define what to measure

SOASTA CloudTest offers unique tools to compare results from different tests that can demonstrate your performance according to your specifications.

- Set limits (your application or site limits may already be determined by established application requirements)

If your site is designed to handle a given number of users, you may want to test its performance at that limit as well as under additional stress--measuring server performance and response times, for example.

- Network dynamics

The test case may necessarily include users executed from more than one network location. [Contact Technical Support](#) for more information about using multiple locations in your tests.

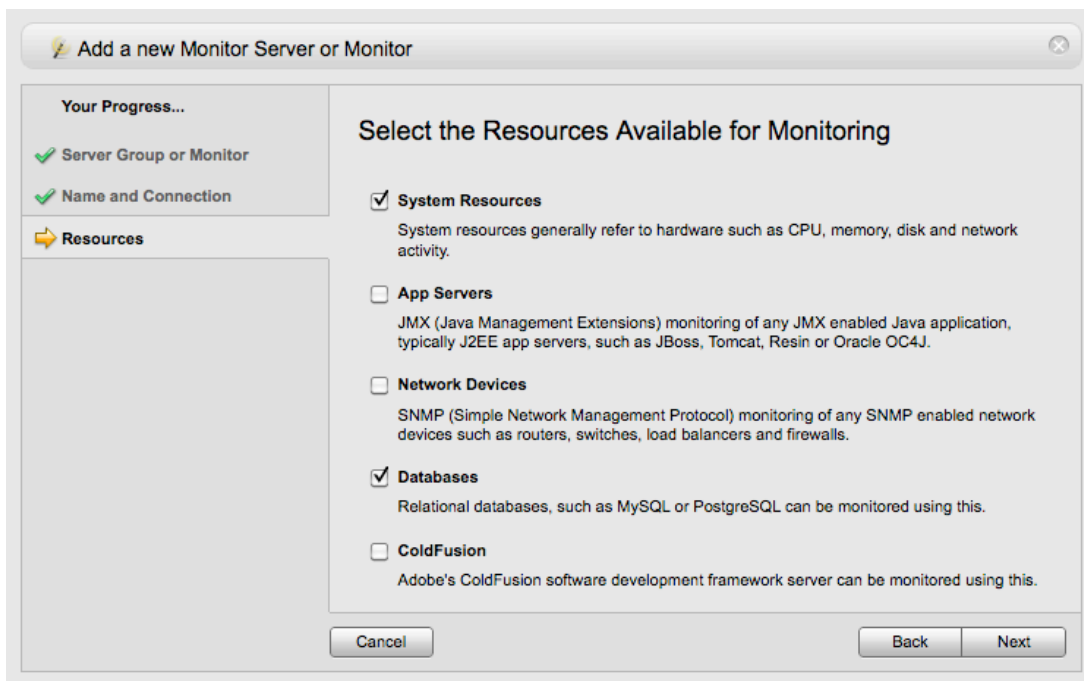
Configuring Performance Monitors

Once you've decided what to measure, you can begin to setup Monitor Servers and Monitors that will provide the analytic details to evaluate your test results.

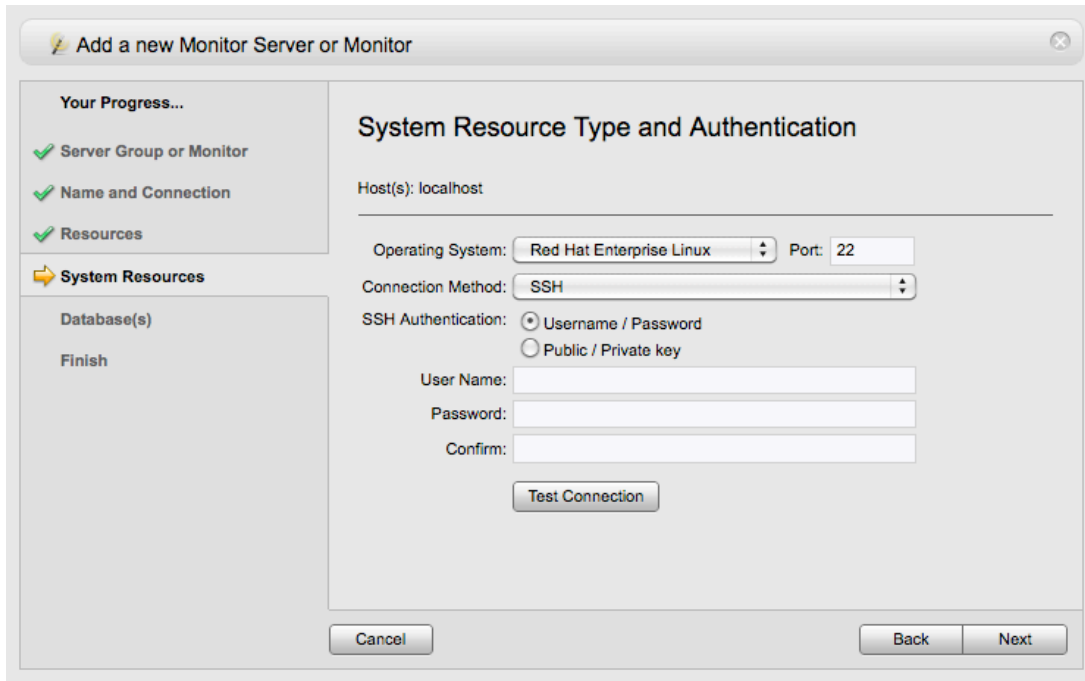
SOASTA CloudTest facilitates monitoring of server- and client-side performance information during the play of test compositions. A Monitor Server defines the hardware or software end-point that will be monitored. Within SOASTA CloudTest, a monitor is an instance of a given Monitor Server. A monitor can use all or a portion of the data offered by its Monitor Server. Once the monitor is configured, it is available to any test composition to provide analytic information about resources.

You can monitor System Resources, Application Servers, Databases, as well as the ColdFusion development framework. You can also monitor JMX and all custom counters from the Windows Performance Monitor.

During monitor creation or editing, look closely at the resources that you want to monitor for the given Monitor Server. Refer to the [Creating Monitor Server Groups](#) and [Creating a New Monitor](#) topics for full monitor configuration instructions.

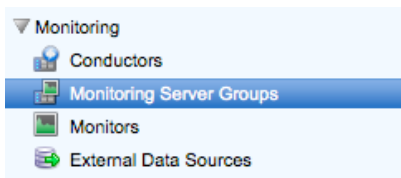


Setting up a Monitor Server requires the hostname and authentication credentials.

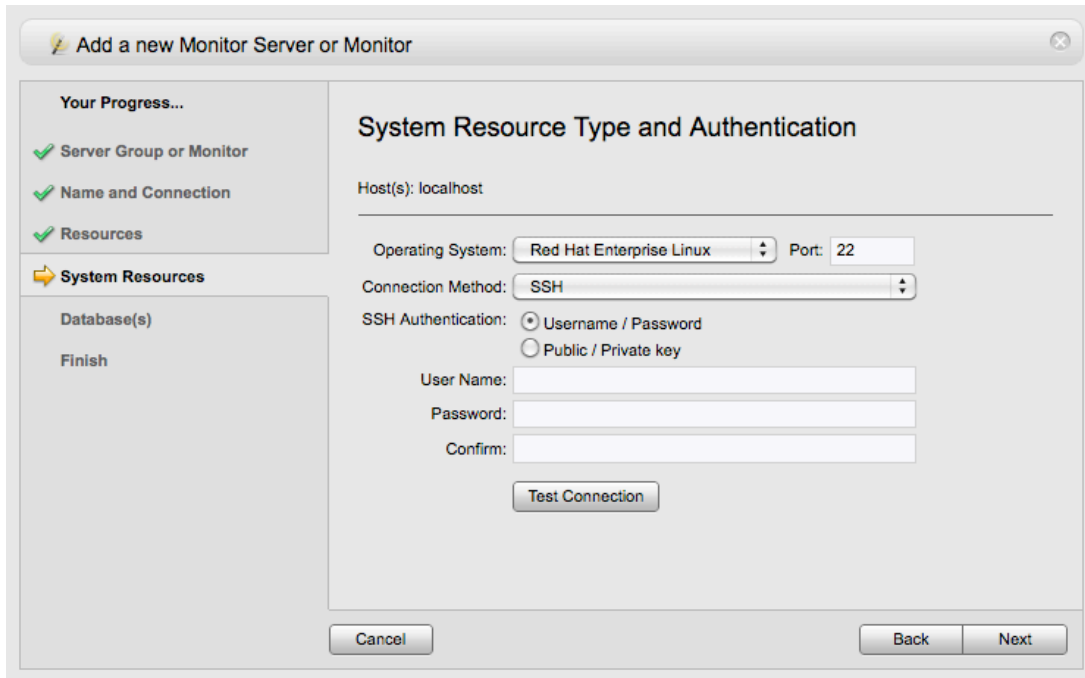


Create a New Monitor Server and Monitor

1. In SOASTA Central, click Central > Monitoring Server Groups.



2. Click New on the toolbar.
3. The Add a New Monitor Server or Monitor wizard appears. Click Monitor Server, and then click Next.
4. Enter the Monitor Server name, a brief description, and the host name of the server.
5. Choose which resources on the host to monitor. You can monitor System Resources, Application Servers, Databases, and ColdFusion development framework. For example, if you want to monitor a JBoss server, check the App Servers box.
6. Enter the authentication information about the monitor server.

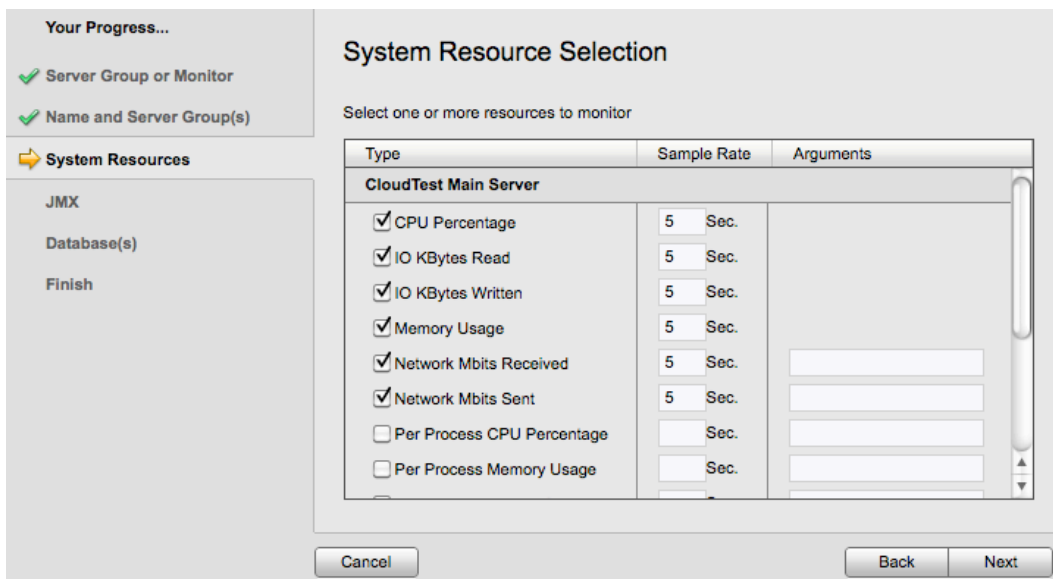


- Once a Monitor Server is configured, you should also create a monitor that uses the newly configured Monitor Server.

To do so, click Next and on the Summary screen check the Create a Monitor for this Monitor Server, or leave the box unchecked and use the Central > Monitors > New command to create a monitor at a later time.

- In either case, click Finish to create the monitor.

TIP: The best approach is to set up the Monitor Server for all its potential capabilities. Then, each Monitor based on it can be as narrow or wide-ranging as necessary.



In the following sections, we will show the Response Time (from the site being tested), as well as CPU percentage of the machine serving SOASTA CloudTest—two very simple performance metrics—for several tiers of users (for example, 100 and 1000 respectively). By multi-selecting several different test results, you can easily view performance as the number of users for a test increases.

IMPORTANT: If you're only practicing, we recommend you don't use the virtual user and multiple location features on any site other than your own. Configuring large tests against public web sites can easily be mistaken for a Denial of Service (DoS) attack.

Composing a Performance Test

A performance test is built using the same SOASTA CloudTest steps as any other test; however, there are additional steps to select the performance monitors configured in the previous steps, to refine logging, and to define the way timeouts are handled. In the following sections, we will use recordings from the sample pages to create a test clip, add it to a test composition, add virtual users, and then refine test composition properties for performance.

Once we have a test that is successful with a low-end benchmark, we will increase the number of virtual users and compare those results to the benchmark results.

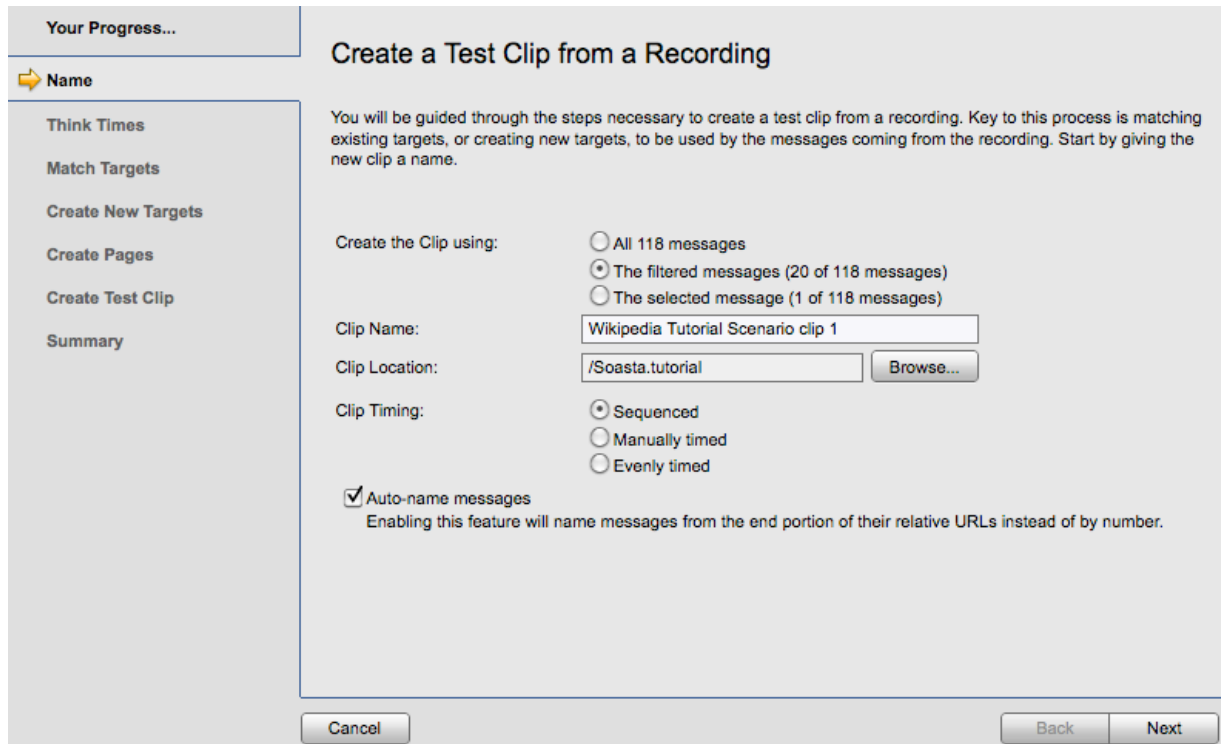
Creating the Test Clip

The following exercise uses existing HTTP(S) message recordings created using Wikipedia. The recordings are converted to the test clip format and can then be opened in the SOASTA CloudTest Composition Editor, or in cases where additional editing of the test is preferred, in the SOASTA CloudTest Clip Editor.

You can use any HTTP(S) recording to create your own test using these same steps. After recording conversion to the test clip format, there are additional performance steps in the Composition Properties tab.

Note: If you don't have an HTTP(S) recording to use, refer to [Creating an HTTP\(S\) Recording](#).

1. Open the recording that you want to convert to a test clip. To do so, select Central > HTTP(S) Message Recordings, select the recording and double-click.
2. Once the recording is open, determine if you want to use all of the messages or a selection of the recorded messages by applying filters before converting.



- To convert all the messages into the test clip, simply click Convert to a Clip on the upper-right.
- Otherwise, apply filters, or select the messages you want to convert and click Convert to a Clip.

In either case, the Create a Test Clip from Recording Wizard appears.

3. Give the test clip a name; select a clip timing mode (Sequenced is the preferred method for the performance context), and then click Next.

- **Sequenced**

In a Sequenced clip, CloudTest sends request #1 in a test clip and request #2 isn't sent until a response to #1 is returned. This is the default selection for test clips created in SOASTA CloudTest and the clip type used by most SOASTA customers.

- **Manually Timed**

In a Manually Timed clip, you can specify down to the millisecond (ms) when a specific message is sent. Responses are not required before the next request is sent.

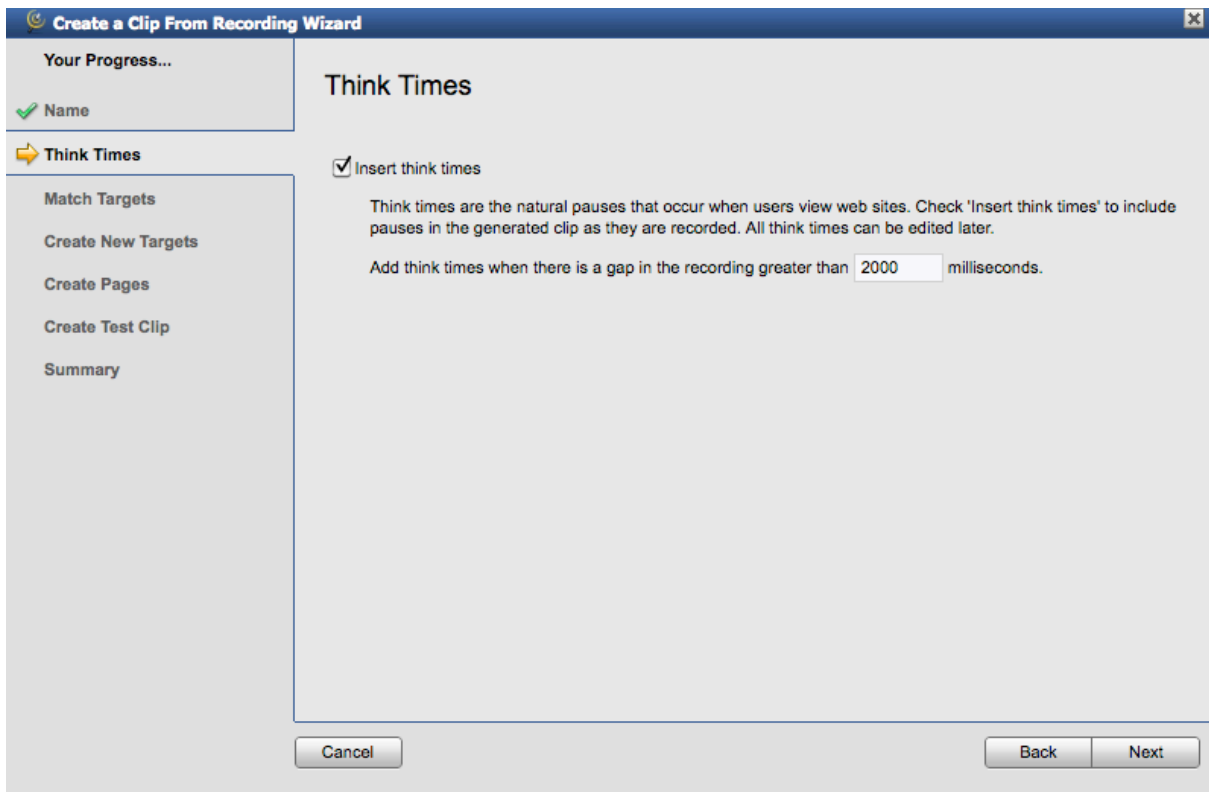
For Manually Timed clips, an additional option to "Detect and insert bursts" appears. In the context of recording conversion, "bursting" is a series of messages that are probably due to a single event. For instance, a web

page that loaded with 17 requests (for images, CSS, JavaScript files). Bursting isn't applicable for non-manual timing types.

- **Evenly Timed**

In an Evenly Timed clip, whenever the test clip is resized in the SOASTA CloudTest Composition Editor, the test clip is re-timed (1 per second, 2 per second). The default is one message sent per second. Responses are not required before the next request is sent.

4. Optionally, set a value (or accept the default) in the Insert Think Times tab.

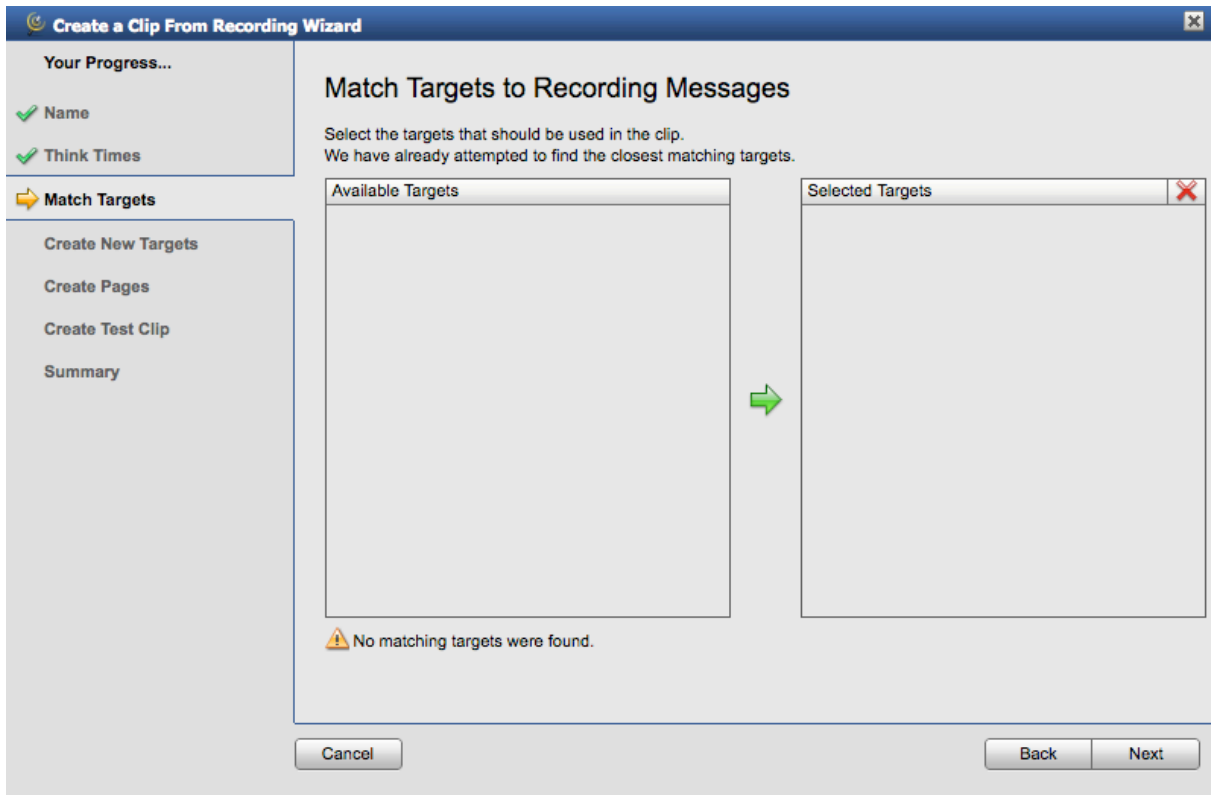


Think times simulate the time that users spend viewing the content of a given page and help to better simulate those time gaps. Whenever a user-configurable gap is detected a “think time” will be inserted within the Test Clip. This will be represented by a time delay clip element in the clip for each “think time”. Think times apply to Sequenced and Manually timed clips only.

If you wish to apply think times, click on the checkbox labeled “Insert think times” and set the threshold values. Pauses will automatically be inserted into the generated clip according to these settings to simulate user think times.

5. In the next step, click the Automatically create missing targets box, and then click Next.

For new users, the Available Targets list will appear empty and a status message appears at the bottom of this list to indicate that no matches (with existing targets) were found.



- If any existing matching targets are detected, they are listed in the Selected Targets list on the right. If the targets do not exist, they will be created in the next step.

- Click Next. The Create New Targets step appears. This step allows new targets to be reviewed one by one, and individually disabled (if a target is unchecked, the related target and its messages won't be created).

Your Progress...

- ✓ Name
- ✓ Think Times
- ✓ Match Targets
- 👉 **Create New Targets**
- Create Pages
- Create Test Clip
- Summary

Create New Targets

New Targets

Create	Target Name	(Hover to see the complete target URL)
<input checked="" type="checkbox"/>	www.wikipedia.org	
<input checked="" type="checkbox"/>	wikipedia.org	
<input checked="" type="checkbox"/>	en.wikipedia.org	

Potential SOAP Targets

Create	Target Name
No SOAP targets need to be created.	

Binary SOAP Endpoints (must enter a WSDL URL)

Create	Target Name
No binary SOAP targets need to be created.	

WSDL URL:

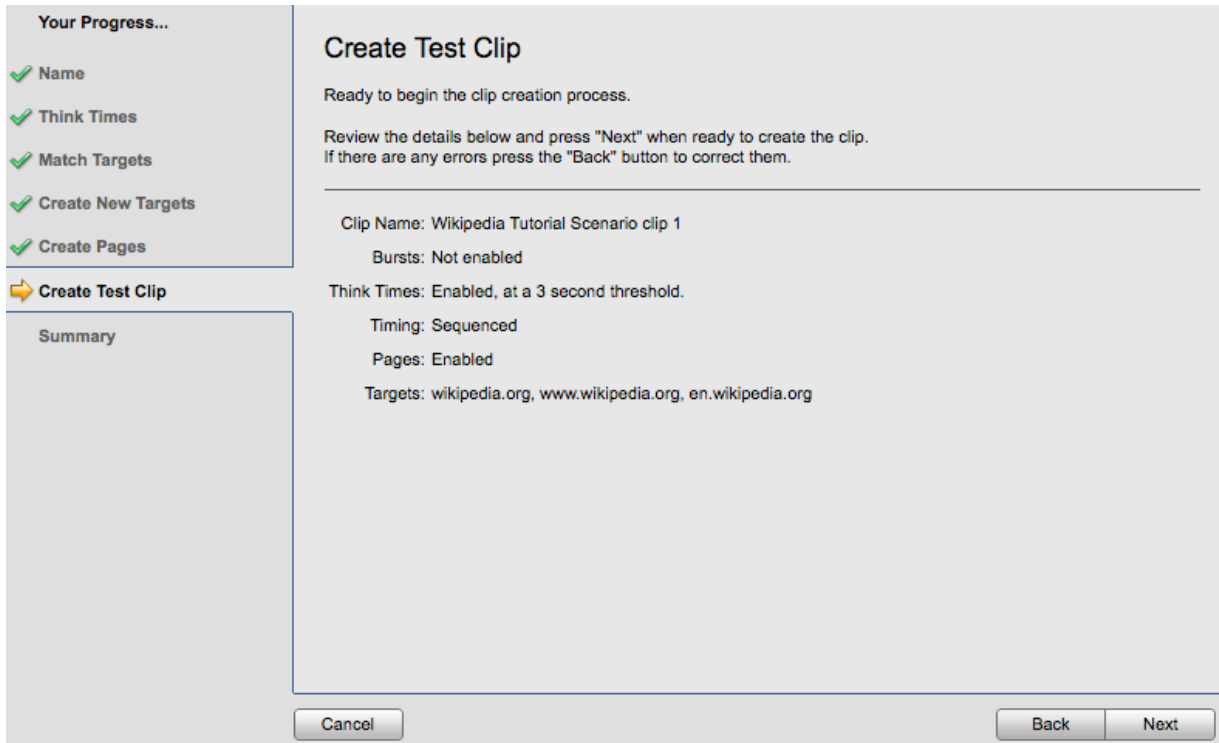
- Click Next. The Create Pages step appears. To edit the test clip. For the purposes of this tutorial, we will continue using *Continue* next to *Go to the Clip Editor*.

8. Leave *Automatically create groups of Page requests* box checked and also check *Mark the first message in the page to dynamically retrieve its resources*. Click *Next*.

The screenshot displays a wizard interface for creating pages. On the left, a sidebar titled "Your Progress..." lists four steps: "Name", "Think Times", "Match Targets", and "Create New Targets", each with a green checkmark. Below these is the current step, "Create Pages", indicated by a yellow arrow icon. Under "Create Pages", there are two sub-items: "Create Test Clip" and "Summary". The main content area is titled "Create Pages" and contains three checked options: "Automatically create groups of Page requests", "Mark the first message in the page to dynamically retrieve its resources", and "Remove the 'static' resource messages from the page". At the bottom of the window, there are three buttons: "Cancel", "Back", and "Next".

The Create Test Clip step appears with summary information about the new test clip selections.

In the example below, the clip name to create is *Wikipedia Tutorial Scenario clip*, think times are enabled at a three second threshold, timing is sequenced, pages are disabled, and two targets to create are listed.

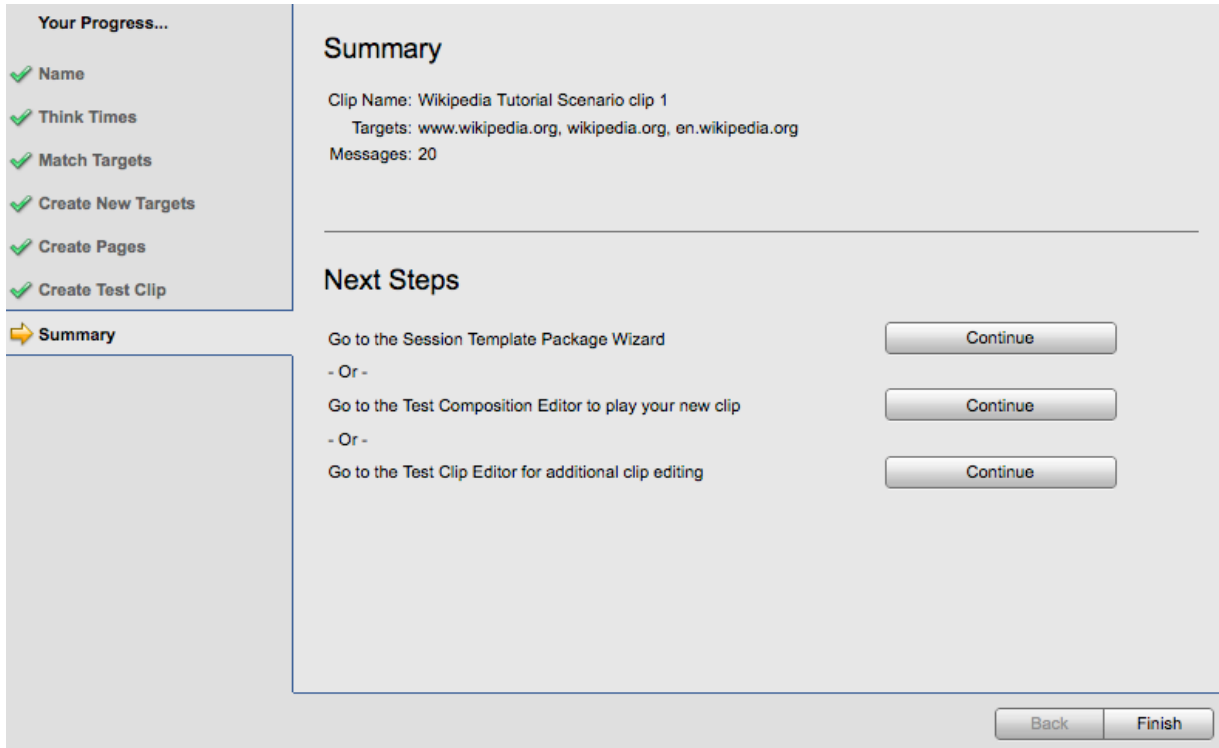


9. Click Next. The status indicator appears to track clip creation progress.
10. The wizard creates the clip and then displays the Summary page. Click Go to the Test Clip Editor to perform additional clip editing tasks, such as user-specific parameterization.
11. For the purposes of this tutorial, we will continue using *Continue* next to *Go to the Clip Editor*.

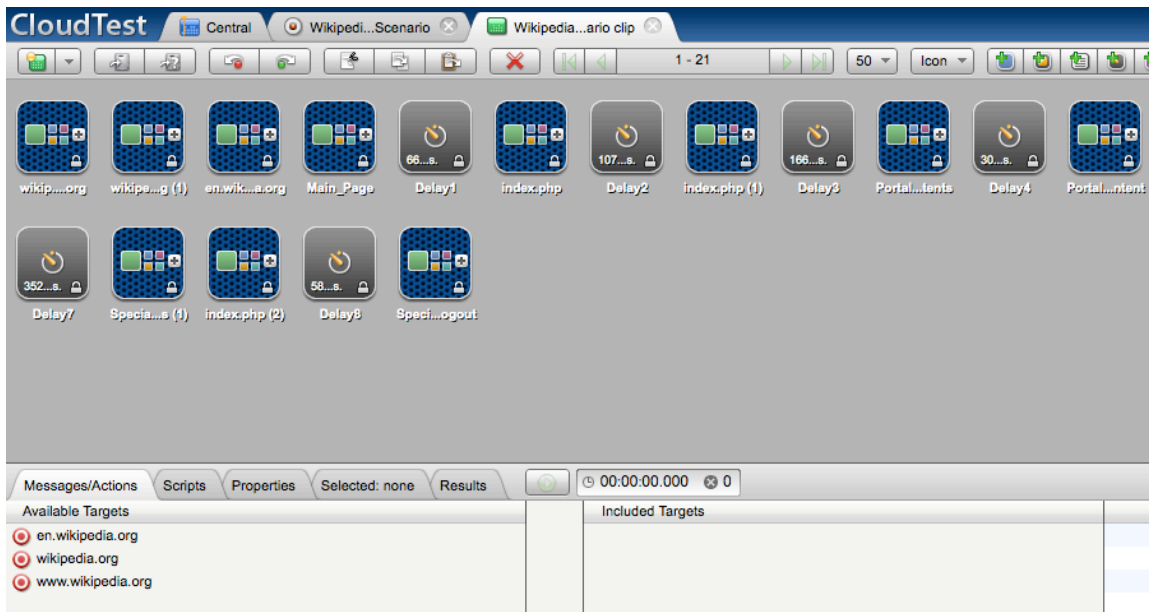
Click *Continue* next to *Go to the Composition Editor* **if you do not wish to edit the test clip** before running your test.

--Or--

Click *Continue* next to *Go to the Clip Editor*, **if you'd like to edit the test clip before adding it to a test composition**. When you do so, the new test clip is opened in the editor. In the remaining steps in this section, we will briefly discuss some useful edits you can make to enhance the readability of your test results.



12. The test clip is opened in the Clip Editor. The pages (and any other clip elements) are displayed in the center workspace, while the lower panel displays the default Messages/Actions tab with Available Targets and the Included Targets and Operations (not shown) lists to the right.



The messages, most of which are within pages in the new test clip, were created from the recorded messages that were selected for conversion in the steps above, while delays were calculated from the recorded think times.

- For tests that contain only static requests, such as those without logins, converted test clips are ready for use in tests as is.

For tests that include unique session data (such as a login that creates a session for a given site), the dynamic data values must be replaced before the test can succeed. The additional clip-editing steps below will configure the test clip to extract dynamic session data, replace it with a custom clip property, and then use that property to place updated session data into the test at runtime. Since the example test clip created above includes such data, the clip-editing steps are covered in the following sections.

13. In the Clip Editor, change to List view to quickly edit messages in this test clip.

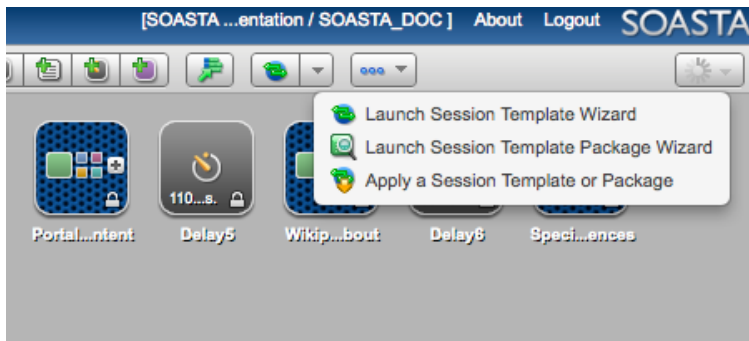
Using the Session Template Package Wizard

Now that a test clip has been created from the recording, the next step is to parameterize those messages that require unique session data.

In this section we will utilize automation techniques via the Session Template Package Wizard, which creates a re-usable “template” that can be applied to other tests that use the same values. These steps can also be done manually using the Clip Editor lower panel.

Parameterization, or correlation, can be achieved by a variety of methods including SubString parsing, XPath location, by creating a new inline script, or by using an existing script.

1. Click the Session Template drop-down menu in the top right of the Clip Editor and then choose the Launch Session Template Package Wizard command.



When you do so, the Session Template Package Wizard appears.

Session Template Package Wizard

Name: Description:

Location: Browse...

Include: Headers Envelope
 POST Data Query String Params Location
 Use custom delimiters Entire URL REST

Exclude: Values shorter than characters
 Names: Accept,Accept-Encoding, Accept-Language, Ca
 Values: true, false, on, off, null, undefined

Session Templates will be created for every selected item. Show Checked Items Only

<input checked="" type="checkbox"/>	Name	Value	Found in Response	Type	Used in Request(s)	Property Name	Value Retrieval Method
-------------------------------------	------	-------	-------------------	------	--------------------	---------------	------------------------

Backup Clip "Wikipedia Tutorial Scenario Automated clip"?

1. First, give the session template a name. For example, *Wikipedia Session Template*.
2. Use the Browse button to specify the *Soasta.tutorial* folder created above as the location.
3. Leave the Include/Exclude fields all checked. For more information about these fields, refer to [Session Template Packages](#).
4. Leave the Backup Clip box on the lower left checked. This ensures that the original test clip will be backed up.

Scanning a Clip for Values to Parameterize

In the following steps, the Clip Scanner will parse the clip in search of matching name/value pairs.

1. Once ready, click Scan Clip.

The Scan Clip button becomes inactive and the scan begins. After the clip is scanned, the scan button automatically becomes re-enabled if the Include or Exclude options are changed. Scan progress is indicated by the green progress bar. Rows begin to appear as the scan progresses.

Once the scan is complete, the results are listed in the table below. Additionally, the green checkmark appears, and the table below is populated with name/value pairs. Some rows may appear before the scan completes.


Session Template Package Wizard

Name: Description:

Location:

Include: Headers Envelope
 POST Data Query String Params Location
 Use custom delimiters Entire URL REST

Exclude: Values shorter than characters
 Names:
 Values:



Clip scan complete. (31 name/value pairs found)
 Session Templates will be created for every selected item.

<input checked="" type="checkbox"/>	Name	Value	Found in Response	Type	Used in Request(s)	Property Name	Value Retrieval Method
<input type="checkbox"/>	Referer	http://www.wikipedia.org/	wikipedia.org	HTML	Main_Page (+1)	Referer	SubString
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Main_Page	Main_Page	HTML	index.php	Referer	XPath
<input type="checkbox"/>	returnto	Main_Page	Main_Page	HTML	index.php	returnto	SubString
<input type="checkbox"/>	wpLoginAttempt	Log+in	index.php	HTML	index.php	wpLoginAttempt	SubString
<input type="checkbox"/>	wpLoginToken	a7ab991719a2d02fc638564bdbfba0da	index.php	HTML	index.php	wpLoginToken	XPath
<input type="checkbox"/>	wpName	Soasta.tutorial0	index.php	HTML	index.php	wpName	XPath
<input type="checkbox"/>	Origin	http://en.wikipedia.org	index.php	HTML	Special:Preferences (+	Origin	SubString
<input type="checkbox"/>	action	submitlogin	index.php	HTML	index.php	action	SubString
<input type="checkbox"/>	type	login	index.php	HTML	index.php	type	SubString
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Portal:Contents	Portal:Contents	HTML	Portal:Featured_conten	Referer	XPath
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Portal:Featured_...	Portal:Featured_con...	HTML	Wikipedia:About	Referer	XPath
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Wikipedia:About	Wikipedia:About	HTML	Special:Preferences	Referer	XPath
<input type="checkbox"/>	wplanguage	en	Special:Preferences	Header	Special:Preferences	wplanguage	SubString

Backup Clip "Wikipedia Tutorial Scenario Automated clip?"

Replace a Token using the SubString Parser

When the first test clip was created, we located the specific message that represented the display of the Wikipedia Log in page in the browser (prior to log in). That message contained the login token, *wpLoginToken*, received from Wikipedia, which was first identified in the recorded response, and then replaced by creating a custom property and assigning it to the selected token.

With Session Templates, this step is now much easier, and additionally, CloudTest will automatically replace all occurrences of the token value with the property path, which is in turn replaced by the actual runtime token when the test is run—without the need to write any JavaScript.

1. In our example, we will replace the highlighted name below. Locate the row for the name `wpLoginToken`.

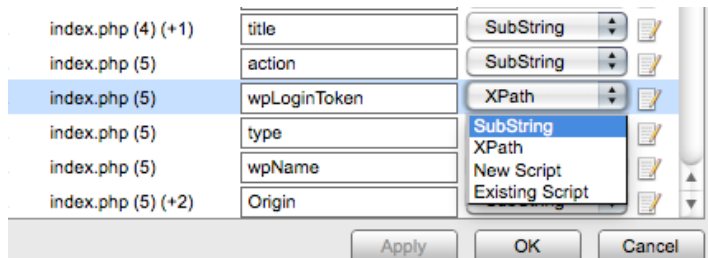
<input checked="" type="checkbox"/>	Name	Value	Found in Response	Type	Used in Request(s)	Property Name	Value Retrieval Method
<input type="checkbox"/>	Referer	http://www.wikipedia.org/	wikipedia.org	HTML	Main_Page (+1)	Referer	SubString
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Main_Page	Main_Page	HTML	index.php	Referer	XPath
<input type="checkbox"/>	returnto	Main_Page	Main_Page	HTML	index.php	returnto	SubString
<input type="checkbox"/>	wpLoginAttempt	Log+in	index.php	HTML	index.php	wpLoginAttempt	SubString
<input type="checkbox"/>	wpLoginToken	a7ab991719a2d02fc638564bdbfba0da	index.php	HTML	index.php	wpLoginToken	XPath
<input type="checkbox"/>	wpName	Soasta.tutorial0	index.php	HTML	index.php	wpName	XPath
<input type="checkbox"/>	Origin	http://en.wikipedia.org	index.php	HTML	Special:Preferences (+	Origin	SubString
<input type="checkbox"/>	action	submitlogin	index.php	HTML	index.php	action	SubString
<input type="checkbox"/>	type	login	index.php	HTML	index.php	type	SubString
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Portal:Contents	Portal:Contents	HTML	Portal:Featured_conter	Referer	XPath
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Portal:Featured_...	Portal:Featured_con...	HTML	Wikipedia:About	Referer	XPath
<input type="checkbox"/>	Referer	http://en.wikipedia.org/wiki/Wikipedia:About	Wikipedia:About	HTML	Special:Preferences	Referer	XPath
<input type="checkbox"/>	wplanguage	en	Special:Preferences	Header	Special:Preferences	wplanguage	SubString

Backup Clip "Wikipedia Tutorial Scenario Automated clip"? Apply OK Cancel

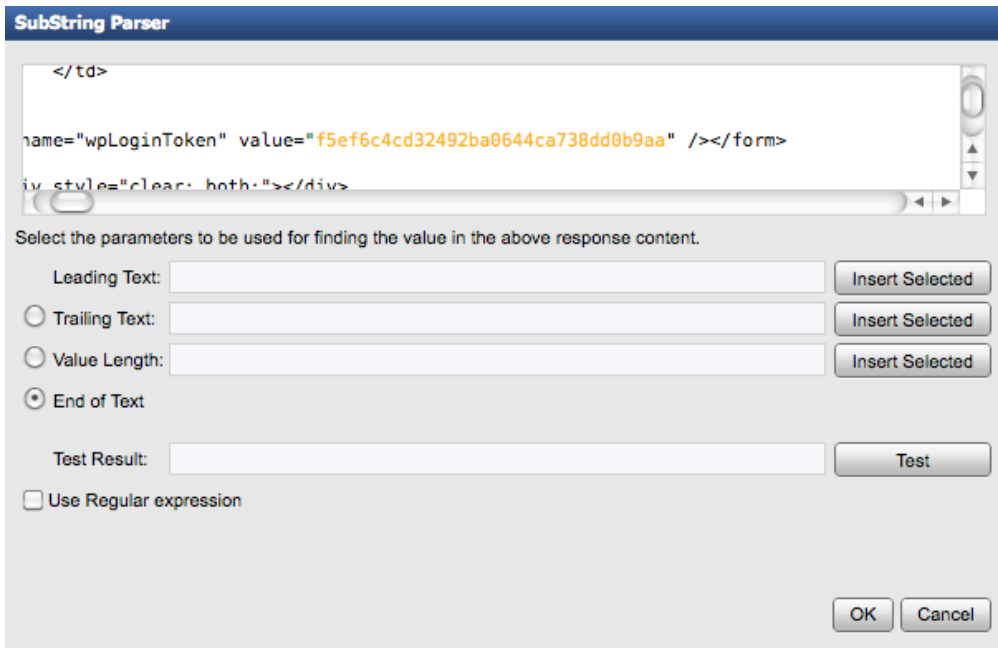
Note the suggested Value Retrieval Method, which in this case is XPath.

Note: XML Path Language, or XPath, is a method for selection within a document. Using XPath can be challenging in cases where a URL frequently changes, since the slightest change to the HTML code for that URL will render the XPath obsolete. SOASTA recommends using SubString whenever possible and XPath only as a last resort.

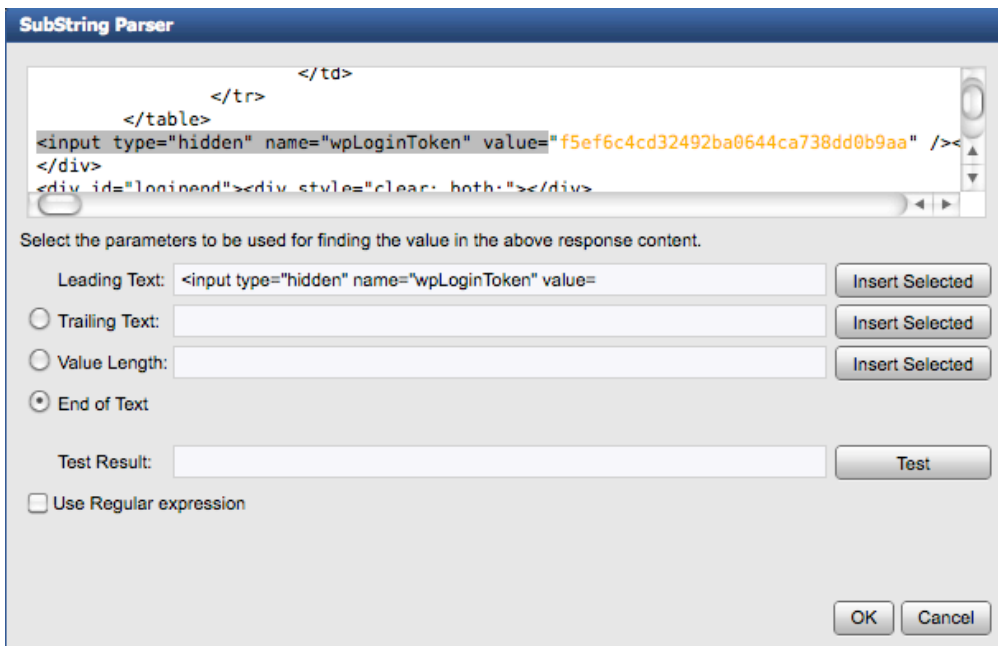
2. Click the Value Retrieval Method drop-down and change it to SubString.



When you do so, the SubString Parser dialog box appears with the token value of *wpName* highlighted in orange in the top section.



2. Use the mouse to highlight the entire string to the left of the highlighted token value and then click the Insert Selected button on the Leading Text row. Be sure to include the quotation mark before the highlighted text. When you do so, the selected string is inserted into the Leading Text field as shown below.



3. Scroll to the right in the top section and use the mouse to highlight the entire string to the right of the highlighted token value and then click the Insert Selected

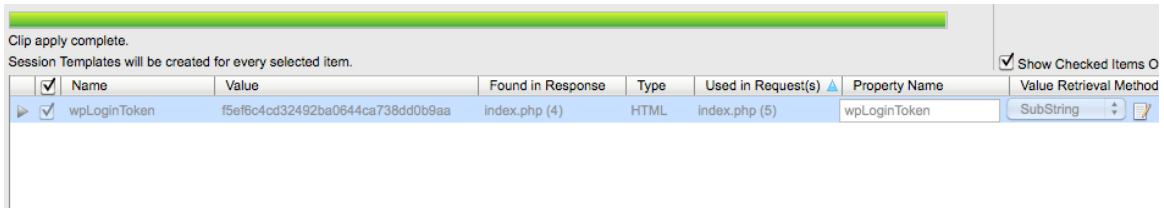
button on the Trailing Text row. When you do so the selected string is inserted into the Trailing Text field as shown below.



4. Click the Test button and ensure that the Test Result shown matches that of the highlighted token value as shown below.



- Click OK to exit the SubString Parser. Note that the wpLoginToken row is now checked and that the specified value retrieval method is SubString Parser.
- In the Session Template Package Wizard, click Apply. The wizard filters the list to only checked items only and applies those values. The green status bar indicates progress and once completed, the green checkmark icon appears.

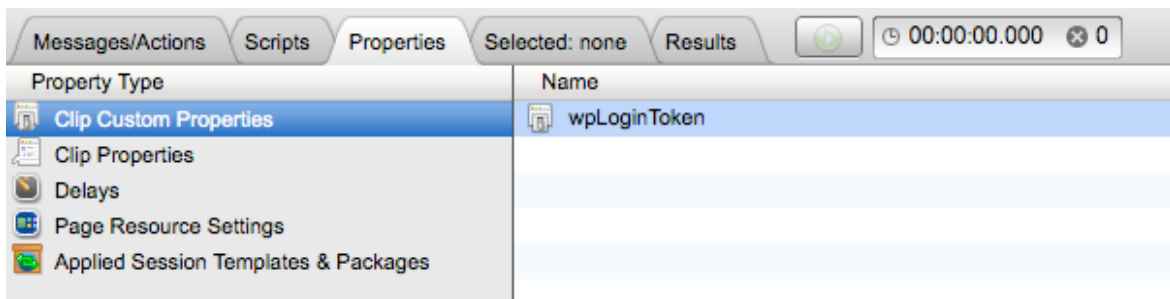


- Click OK to exit the Session Template Package Wizard.

Inspecting Applied Session Template Values

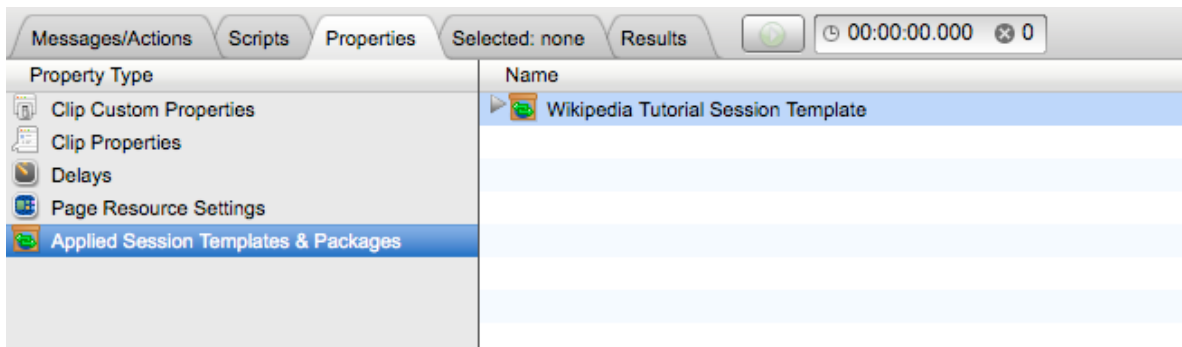
Now that the *wpLoginToken* value has had a template applied, let's take a look at the test clip to inspect the applied template value.

- Click the Properties tab in the lower panel of the Clip Editor

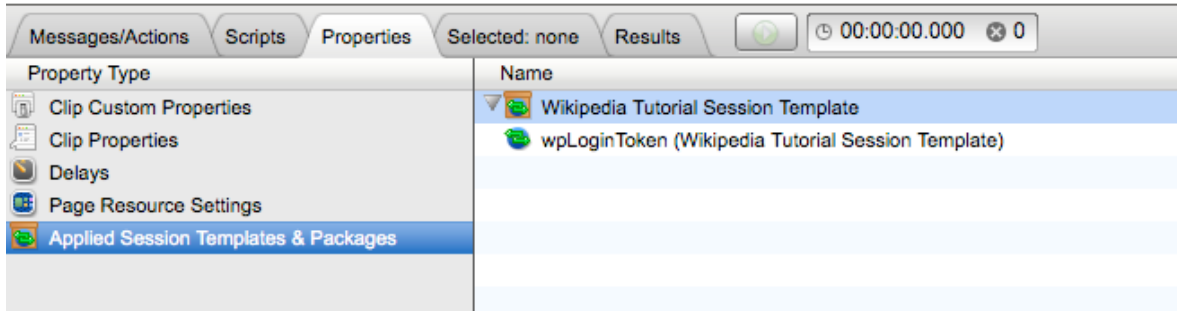


The custom property has assumed the name extracted from the Wikipedia site, which was identified by the SubString Parser steps above.

- Click the Applied Session Templates & Packages node in the Property Type list.

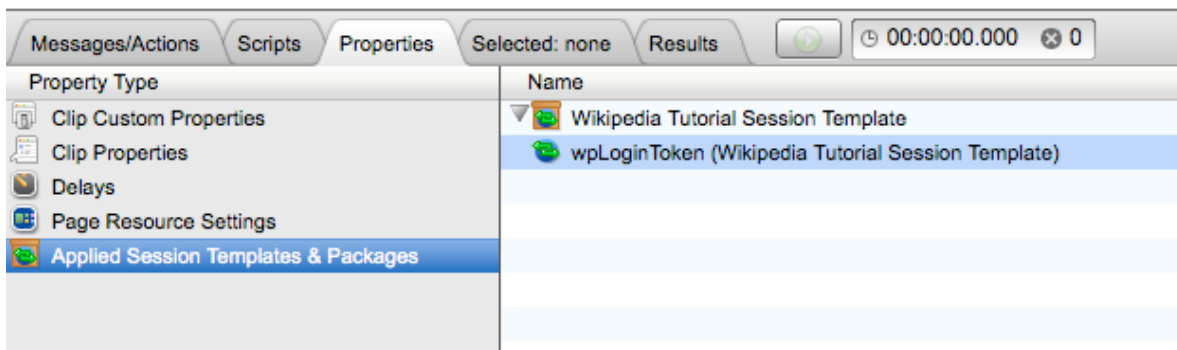


- The Wikipedia Tutorial Session Template appears in the Name list on the right. Click its arrow to expand it.



Note that the session template names correspond to the fields upon which they were based and also correspond to the automatically created custom clip properties.

4. Click the wpLoginToken item in the expanded session template.



5. Review its Summary details on the far right pane. Note that the retrieval variable value method, the leading text, and trailing text match the methods and selections used above.

Summary

Applied by: SOASTA_DOC on Mon May 16 06:39:15 PDT 2011

Retrieve variable value method: Substring
 Leading text: <input type="hidden" name="wpLoginToken" value="
 Trailing text: " /></form>

Message extracted from: ■ index.php (4) Highlight

Message(s) applied with ISSEs: ■ index.php (5)

Property Path: wpLoginToken
 Path Type: clip

The Summary identifies the first message from which the value was extracted and also any subsequent messages to which the value was applied by the use of a property path (or ISSE).

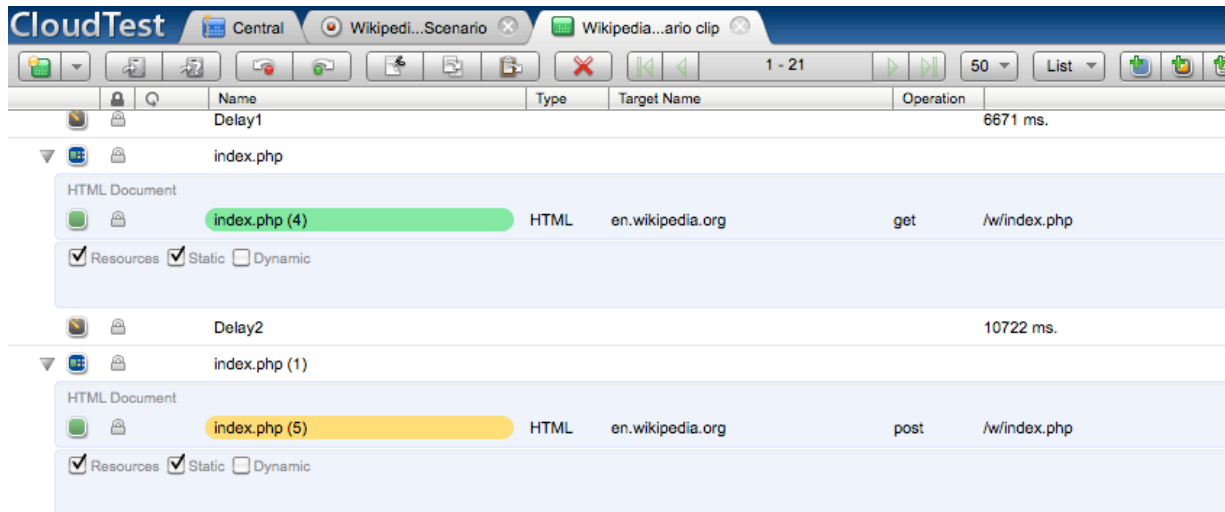
For messages with applied ISSE values, a property path was used to parameterize a value. The given path refers to the property that will insert the runtime value into place during the test. See [In Situ Substitute Expressions \(ISSE\)](#) for more about using paths to refer to custom or global properties.

6. Click Highlight. CloudTest changes the workspace display to show those messages that the value was extracted from and to which it was applied.

In List view, pages appear as shown below

Note: CloudTest has three collection types: pages, groups, and chains. For more information about collections, see [Viewing Collections](#).

The HTML Document row for a given page is the CloudTest equivalent of the Wikipedia page created in the recording above. This row is wider and contains three checkboxes: Resources (checked), Static (checked), and Dynamic (unchecked).

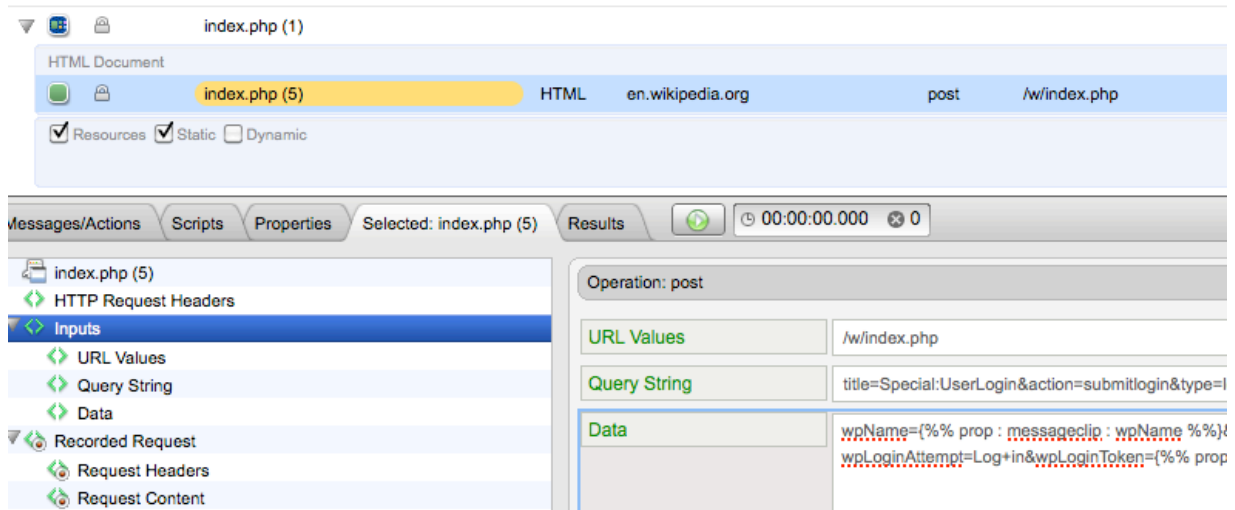


The first page shown above, *index.php*, contains the message highlighted in green, *index.php (4)*. This message is the “HTML Document” (or main message) of its page. The *index.php (4)* message, and the *wpLoginToken* value extracted from it, originated from the click that opened the Log in page (prior to submitting the log in).

The second page shown above, *index.php (1)*, contains the message highlighted in yellow, *index.php (5)*. This message is the HTML Document of its page.

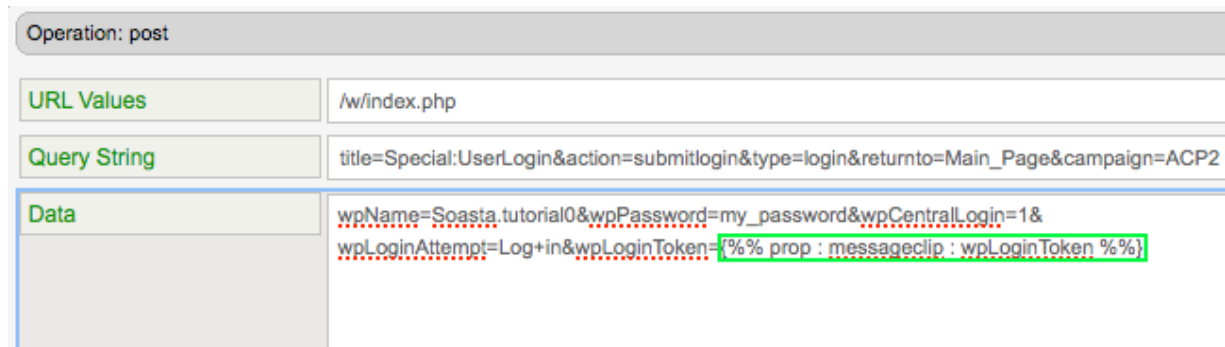
The *index.php (5)* message is a POST message that sends back the user name, password, and the required *wpLoginToken*. Without all three the login fails.

7. Double click the POST message in the expanded page to open it in the lower panel. This message is also the main HTML Document within the page.

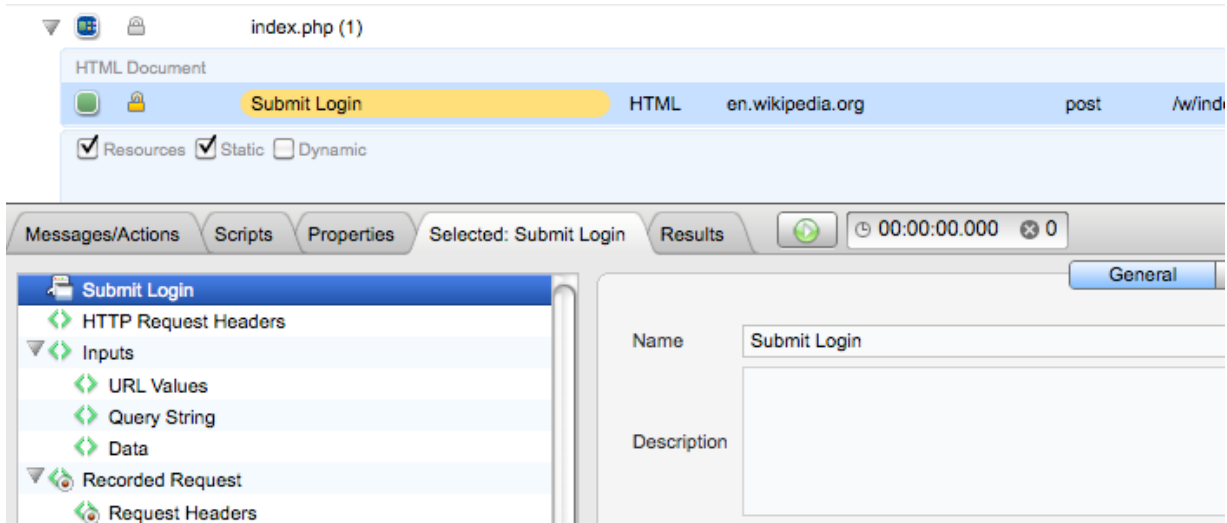


The message opens in a new tab in the lower panel (e.g. *Selected: index.php (5)*).

- Verify that the wizard replaced the login token with the property path for *wpLoginToken* (shown below).



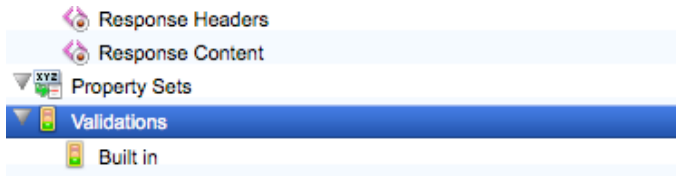
9. Once the message has been examined, rename it to *Submit Login* by selecting the top node (where the message name appears) and entering the new name. Renaming individual messages helps to identify them easier in test results.



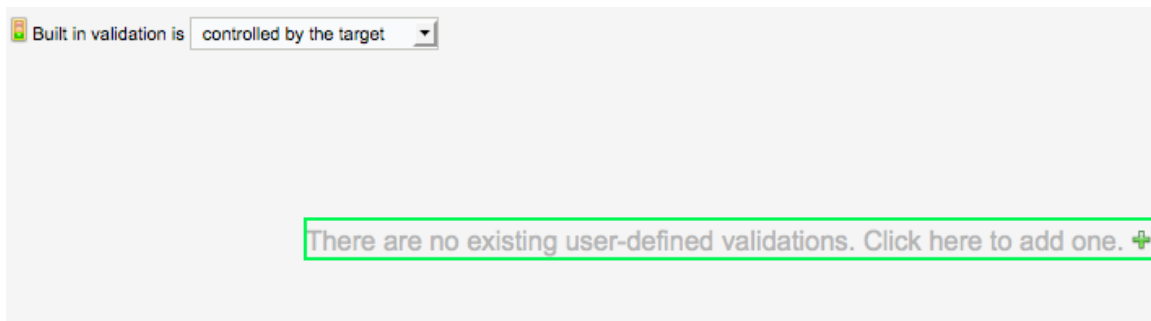
Creating a User-Defined Validation

CloudTest provides user-defined custom validations. Use the following steps to define a custom validation that will validate login success in the test clip.

1. Click the message icon for the newly renamed *Log in* message a second time, select the Validations node in the list on the left.



2. Click the green arrow in the field on the right to add a user-defined validation.



The Validation form appears. Accept the default value, *This validation: validates a portion of the response body.*

This validation:	validates a portion of the response body
XPath:	
Response Type:	XML
Match type:	Matches exactly
Constant expression	
Failure Action:	Fail the parent

[Customize result success/error messages](#)

3. Set the Response Type to *Text* and the Match type to *Matches glob expression*.

Match type:	Matches glob expression
Constant expression	*Login successful*
Failure Action:	Record in results only

[Customize result success/error messages](#)

4. Enter the Constant Expression, **Login successful**.

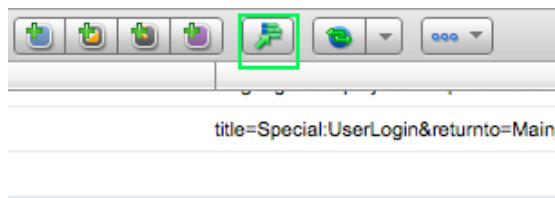
5. Click Save in the Clip Editor toolbar.

Playing and Debugging a Simple Test

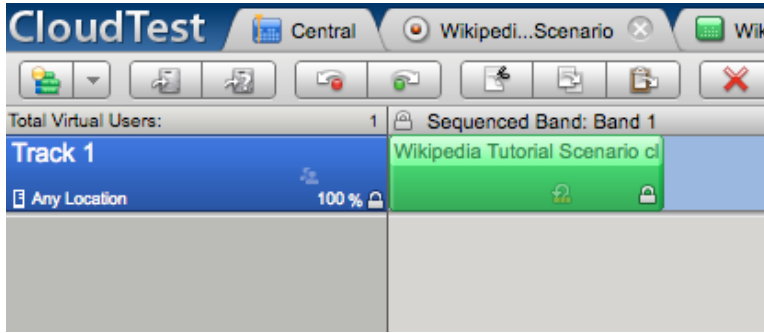
As a general guideline, it is best to play and debug a test *as components are added*. This simplifies debugging tasks.

Additionally, this type of debugging is done before adding virtual users (covered in the final sections) and while the test is small enough to be run with the default verbose logging (e.g General play mode) that best utilizes the Result Details dashboard.

1. Click the Use in Test Composition icon on the Clip Editor toolbar and then select Open in Test Composition.



The test is added to Track 1 of a composition with the name “Composition for [Name of Test Clip]”, which opens in the Composition Editor (in a new tab). For example, the *Composition for Wikipedia Tutorial Scenario clip 1*.



Click Save on the Composition Editor toolbar. The Save Composition box appears. Assign the composition a new name. For example, *Wikipedia Tutorial Scenario composition 1*.

- Click Play on the Composition Editor toolbar. The Composition Editor, Play tab appears with the default dashboard in display while the test composition plays. The Result Details dashboard appears with the status *Playing*.

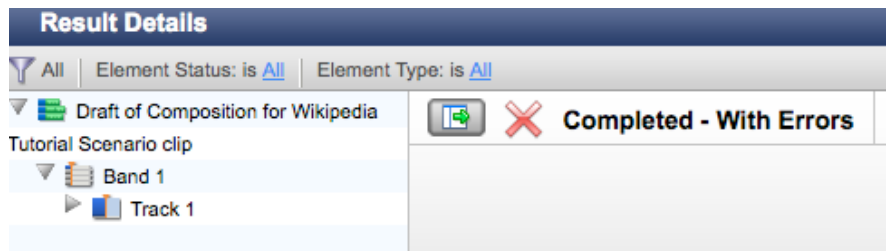
Result Details Dashboard

The Result Details dashboard helps you determine the cause of errors in your test and has several methods for navigating through the test results.

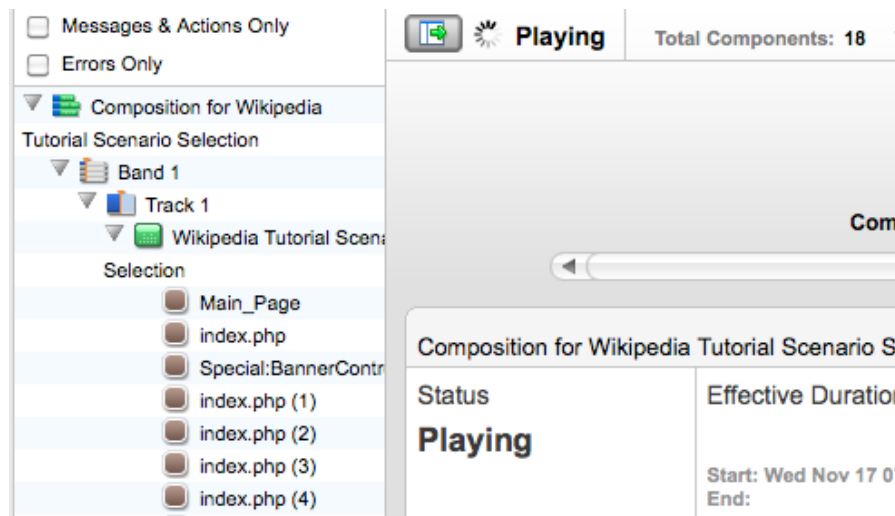
Status	Effective Duration	Avg. Response Time	Total Message Bytes
Playing	Start: Fri Nov 05 12:45:20 GMT-05:00 2010 End:	Min: Max:	S: R:

Component Summary	Retries due to unavailable local ports	Retry time due to
Messages	Maximum	Maximum
Browser Actions	Minimum	Minimum
Scripts	Average	Average
Delays	Total	Total
Checkpoints		
Events	20	
Error Events	0	
Error Components	0	

Whenever errors occur, the Result Details widget helps to easily identify both where and what errors occurred. Select items in either the Cover Flow at the top or the Navigation tree on the left.



The Navigation Tree view on the left is useful for navigating around multiple clips in a test composition. Click the arrow for any node to expand or collapse it.

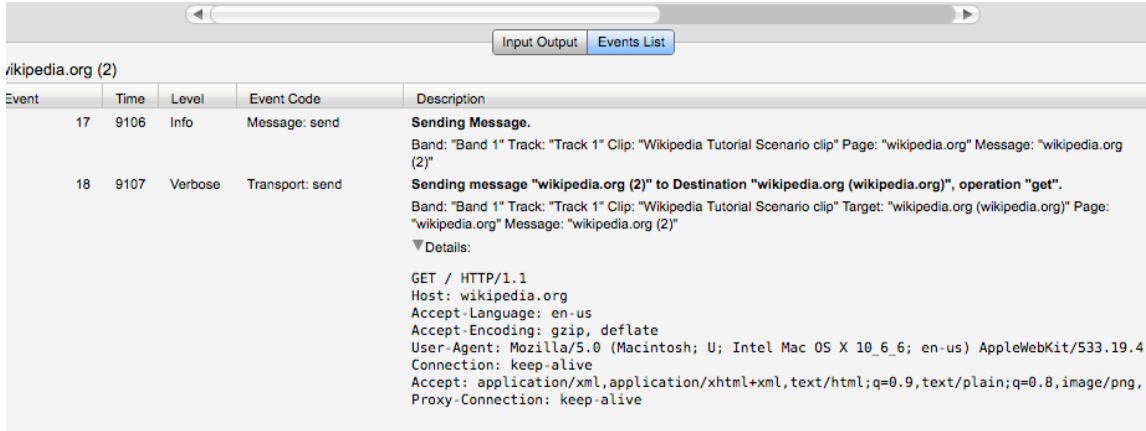


Use the filters to quickly display messages (or browser actions) only (within a single band, track, test clip, or chain); or, to display errors only.



- Click any object in the Cover Flow at the top to center it and display its request/response details and play statistics in the panes below.
- Use the scrollbar to browse the Cover Flow (if an item is selected in the Navigation Tree, the Cover Flow shows only items that pertain to the selection).
- Select any item in either the Navigation Tree or the Cover Flow to show its low-level details in the Input/Output or Events List tabs on the right.

- Click the Events List tab to see a sequential, event-by-event list of the given test result. The Events List displays the full request headers/POST data and response headers/content in the same view. This list provides the raw response data including hidden fields.



Verifying Test Parameterization

Login success means that all the above settings and a test without error means, in general, that the extraction of the wpLoginToken and its runtime replacement worked. Of course, the remainder of the test must also run without error. In terms of parameterization, we can verify that:

- Login succeeded (shown above; since no parameterization has been done in the test as yet we will revisit this issue in the final section below via use of the i script in combination with virtual users)

1. With the Result Details Dashboard in display, expand the Navigation Tree until the POST message is visible in the clip. Once the POST message is in display, select it to determine whether the login was successful.

The screenshot shows a web performance tool interface. At the top, it indicates 'Completed - With No Errors' and provides summary statistics: 'Total Components: 45', 'Total Messages and Actions: 28', and 'Error Components: 0'. The main content area displays a 'Submit Login' button with a 'POST' label. Below this, a breadcrumb trail reads 'Band 1 > Track 1 > Wikipedia Tutorial Scenario - Find and Replaced > Submit Login'. There are tabs for 'Input Output' and 'Events List'. A table summarizes the request details:

Start Time	Delta from sched	Response Time	Total Time	Total Message Bytes
18 sec.		279 ms. TTFB: 207 ms. TTLB: 278 ms.	282 ms.	S: R(Gzip): 8556,404

Below the table, there are two tabs: 'Request' and 'Response'. The 'Request' tab shows the following details:

```

Operation: post
POST /w/index.php?title=Special:UserLogin&action=submit
Host: en.wikipedia.org
Content-Length: 110
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-us
Cookie: enwiki_session=13edfd81bf404680c5f87da0a653fbfa
Accept-Encoding: gzip, deflate
Referer: http://en.wikipedia.org/w/index.php?title=Spec
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 1
Origin: http://en.wikipedia.org
  
```

The 'Response' tab shows the following details:

```

Login successful
From Wikipedia, the free encyclopedia
Jump to: navigation, search
You have successfully signed in to
Wikipedia as "Soasta.tutorial1".
  
```

Scroll back up in the Response tab and note that a successful message gets the HTTP response code, 200.

The screenshot shows the 'Response' tab with the following details:

```

Response
Target: wikipedia.org
HTTP/1.0 200 OK
Date: Sat, 13 Mar 2010 04:16:57 GMT
Server: Apache
Cache-Control: s-maxage=3600, must-revalidate, max-age=0
Last-Modified: Fri, 12 Mar 2010 21:39:12 GMT
Content-Encoding: gzip
Vary: Accept-Encoding
Content-Length: 12172
Content-Type: text/html; charset=utf-8
X-Cache: MISS from sq61.wikimedia.org
X-Cache-Lookup: HIT from sq61.wikimedia.org:3128
Age: 53
  
```

TIP: While the HTTP 200 response code is desirable, an examination of all pertinent HTTP requests is still required to define a successful test, since receiving all HTTP 200 status codes DOES NOT necessarily mean the test is running properly. Many applications will return an HTTP 200 status code but really be in a failure state. This is true because response content can be an error message in some cases. SOASTA simply highlights HTTP 400 & 500 status codes as “errors”.

- Extraction and replacement of the wpLoginToken worked
2. In the steps above, the *Log in* message (e.g. the message that loaded the Wikipedia log in page) contained the wpLoginToken. We can verify that the substitution worked in the test composition by finding the new token in the subsequent Submit Login request (as shown below).

Start Time	Delta from sched	Response Time	Total Time
21 sec.		4 sec. TTFB: 4 sec. TTLB: 4 sec.	4 sec.

Request

```

e/png, */*;q=0.5

Page&campaign=ACP2

oLoginAttempt=Log+in&wpLoginToken=d0a3c84b3c65f11deaa84cb20b711f92
  
```

- Validation worked
3. With Submit Login still selected, click the Events List tab and scroll down until the *Validation of response body passed* event is in display. Click its Details arrow to display full information for this event.

Event	Time	Level	Event Code	Description
82	26141	Verbose	Transport: sent	Received response to message "Submit Login". Band: "Band 1" Track: "Track 1" Clip: "Wikipedia Tutorial Scenario clip" Target: "en.wikipedia.org (en.wikipedia.org)" Page: "index.php (1)" Message: "Submit Login"
83	26143	Verbose	Validation: vcpass	Validation of response body passed. Band: "Band 1" Track: "Track 1" Clip: "Wikipedia Tutorial Scenario clip" Target: "en.wikipedia.org (en.wikipedia.org)" Page: "index.php (1)" Message: "Submit Login" Details: Glob expression match passed: Expression: *Login successful* Value: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org<html lang="en" dir="ltr" xmlns="http://www.w3.org/1999/xhtml"><head>

Adding Test Complexity

Now that the success of a simple test has been verified, a test of greater complexity can be built. With one or more parameterized test clips, a performance test simulating many users can be built quickly using one or more of the methods below.

In this section, we will utilize the test composition created above.

Applying Virtual Users

SOASTA CloudTest uses parallel repeats per track as the basis for generating virtual users. Parallel repeats fire the track concurrently, or in parallel. Tests of great complexity can also be built quickly using multiple test clips and virtual users.

IMPORTANT: Recording from a host is fairly benign, however, once you add virtual users and run a test against that same host your test can easily be perceived by that host as a Denial of Service (DoS) attack. Please review the SOASTA Terms of Service before setting up tests using virtual users. The example test in this tutorial uses very low numbers of virtual users. SOASTA CloudTest Lite is limited to 100 virtual users. Other CloudTest editions are capable of much higher numbers and tests of great complexity with multiple user scenarios, clips, and tracks.

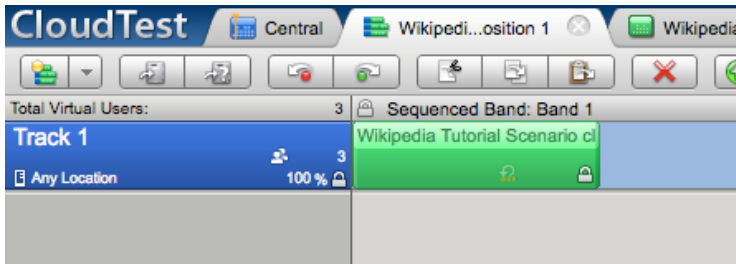
Note that The Composition Editor displays the current number of Total Virtual Users at the top of the Track column.

1. In the Composition Editor, click the blue area on the left labeled Track 1. In the next step, we will set three parallel repeats.
 - Parallel repeating – All of the repeats play simultaneously in parallel. In CloudTest, a virtual user is counted whenever a parallel repeat is set on a track.

Parallel repeating is allowed only for:

- All Tracks.
 - Clips in a Timed Band.
 - Any item inside a Timed Clip that is not inside a Chain or Page.
- Serial repeating – The repeats play serially, one at a time, one after the other. Serial repeating is allowed for *any* item type, although certain item types (such as Checkpoints) only permit the repeat count to be either zero or one.

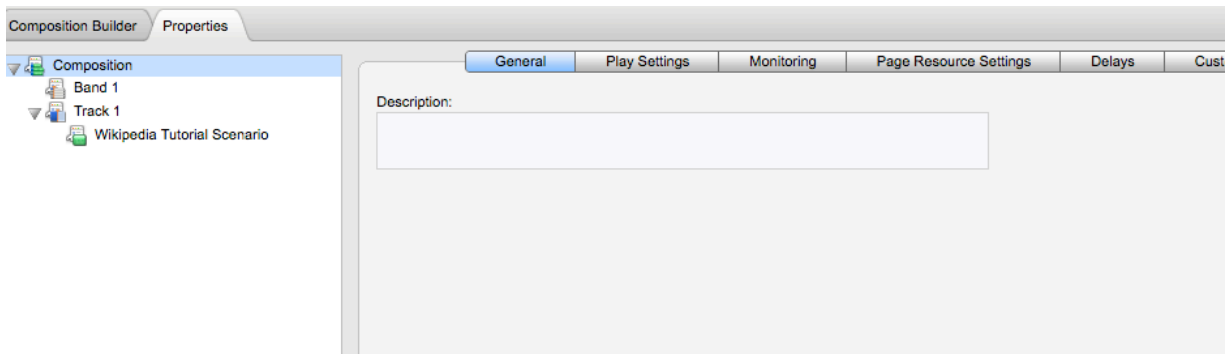
2. Click the Virtual Users icon in Track 1 and enter 3 into the field that appears. Leave the test clip on this track at its default, 1.



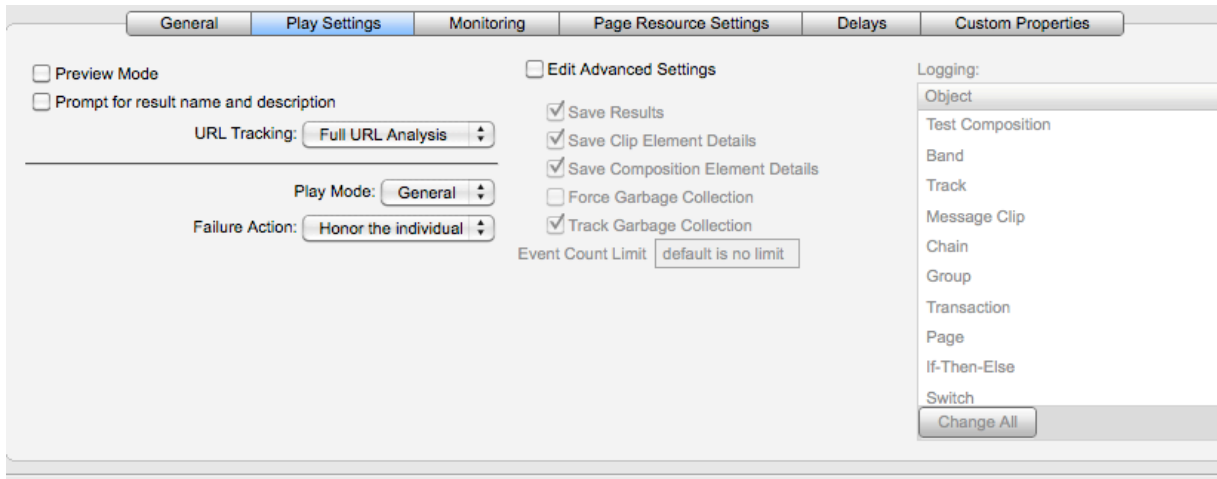
3. Click Save on the Composition Editor toolbar.

Refining General Properties for Performance Testing

1. With the composition open in the Composition Editor, click the Properties tab in the lower pane.
2. Select the Composition node in the Property Type list. The composition's General Properties are shown on the right.



3. Click the Play Settings tab. The default verbose settings (e.g. General mode) are shown below.

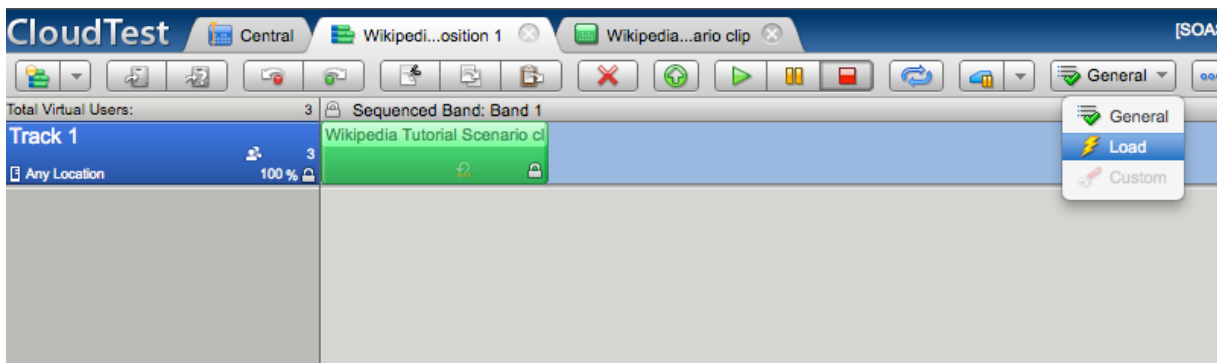


4. In the Failure Action drop-down, select *Never halt on error, regardless of component settings*. In this test, and subsequent performance tests, we want to view errors, such as timeouts, without having the test play stop due to them.
5. Click Save on the Composition Editor toolbar before proceeding.

Refining Logging Properties for Performance Testing

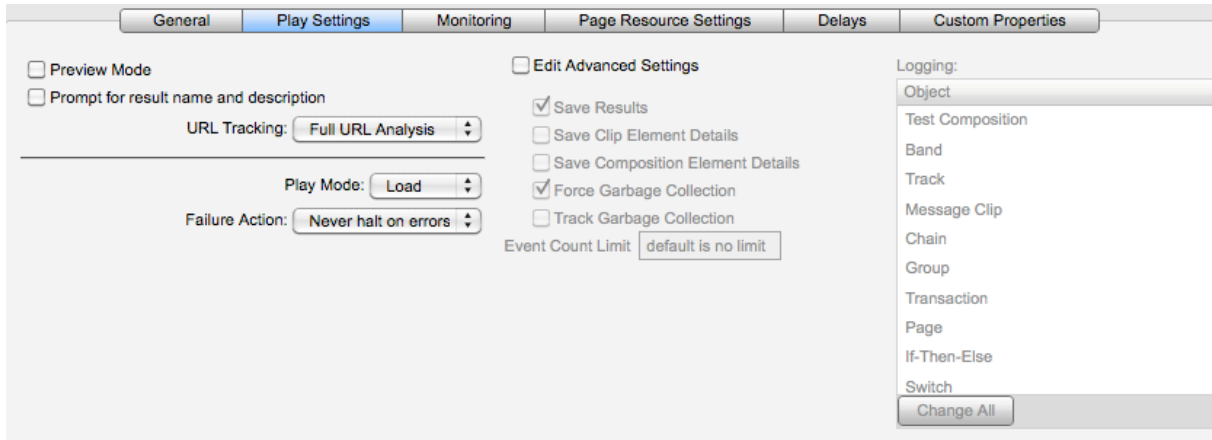
CloudTest provides play modes that comprise "recommended" settings for both debug (General) and load/performance (Load) testing. General play mode provides verbose logging settings that come at a low overhead in smaller tests. Load play mode omits verbose logging, but captures information to help identify performance bottlenecks in a web application.

1. In the Composition Editor, click the Play Mode drop-down and choose Load.



SOASTA CloudTest provides default verbose logging (e.g. General mode) of all its activities that is quite detailed, and in most cases, provides valuable information for little cost. For larger tests with 10 virtual users or more, verbose mode may be a lot more information than is useful for the test.

1. To quickly turn off verbose logging, set Play Mode to "Load." Load Mode settings are shown below.



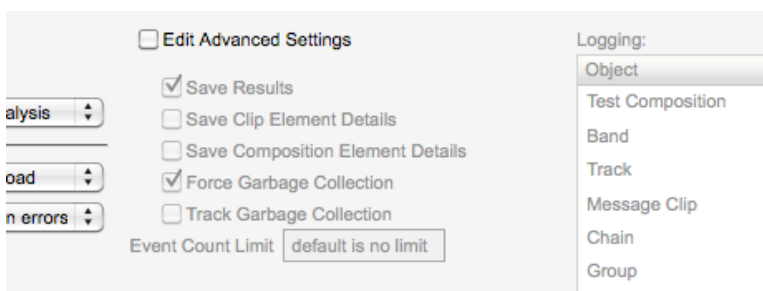
Example Logging Settings for Performance Testing

If Play Mode doesn't provide the desired settings, set Custom Mode (or simply change individual logging settings, which also sets play mode to Custom). SOASTA recommends the following settings for performance (and load) testing:

1. Save Results and Force Garbage Collection checked.
2. For Loggin, Test Composition, *Verbose*. is set
3. For all other logging objects, *None*.
4. Uncheck both Save Clip Element Details and Save Composition Element Details.
5. Click Save in the Composition Editor toolbar after making any changes.

Save Clip Element and Save Composition Details

The Results Service performs a first level of aggregation within the Maestro service, and optionally, only sends aggregates to the Results Service. This has significant performance benefits for large tests, as much less data needs to be passed across the network. However, in the case of a performance test this may not be desirable. Besides network traffic reduction, testing also shows a large benefit in reduced CPU usage.



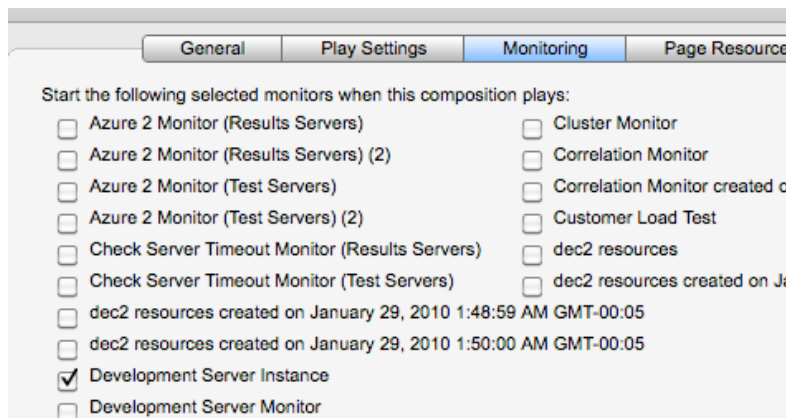
1. To take advantage of this feature, simply uncheck the "Save Clip Element Details" and "Save Composition Element Details" checkboxes in the Composition Editor > Properties > General tab. Note that these boxes are already unchecked in Load Mode (as shown above).

For example, in large tests it is best to turn off details in order to achieve better overall system performance. Turning off details will result in the Results Details widget being empty, but most other widgets and charts will work fine using aggregated data in real time while the test runs. At the very least, be aware of the potential impact of this setting in performance testing.

Monitoring a Performance Test

Now, as a final step in performance test preparation, we will deploy the performance monitor(s) we created above in the current test.

1. With the Composition node still selected in the lower panel Properties tab, click the Monitoring tab.



2. In the Configuring Performance Monitors section above, we create a monitor called Development Server Instance. Check this or any other monitors you'd like to run during test execution.
3. Click Save after making changes.

Adding Additional Clips and Tracks

In our example test, so far we have added one test clip with some additional composition-wide property settings for results, monitors, and with the Save Analysis Details box unchecked.

The test can be saved and played with these settings. However, in reality a much more complicated testing scenario is likely and you will likely want or need to edit your test clips using the Clip Editor via Central > Test Clips.

These are several ways to build a more complex test:

- Create edited test clips using the Clip Editor to add messages, delays, checkpoints, as well as browser actions, as well as to create chains. For more information about test clip editing, see the [Test Editing](#) section of Findouthow.
- Create more than one test clip from a given recording to create user variation
- Create additional recordings with similar, but not identical paths
- Create duplicates of tracks or different sequences of test clips on tracks

If there is only one user case in your scenario (e.g. a user case is usually equated to a test clip), you can duplicate the existing track. Duplicate the track rather than adding virtual users to it. This can be useful if you want to play the test from multiple locations by configuring each track to use a different location.

- Optionally, right-click the highlighted (blue) track area. The context menu includes commands for *Duplicate* and *Duplicate N times...*
- Optionally, add one or more test clips to the test composition by dragging them from those listed in the Composition Editor sub-pane.

Using Locations and Servers (CloudTest Standard and CloudTest Pro Users)

Multiple locations can distribute load among servers or geographic locations or networks. Locations can be assigned

For example, some resources may only be available on certain servers or locations, requiring that a test connect separately to those resources. Or, the test itself may include a requirement to access and measure performance using the cloud from specific geographic locations or networks.

Note: Users without administrative access may only select from pre-configured resources. You can select server resources, such as locations, that are configured by your SOASTA CloudTest Administrator. If the resources you'd like to deploy don't exist or are yet to be configured, discuss this with your administrator or with your SOASTA representative.

Server resources can be selected and set composition-wide or set on a per-track basis.

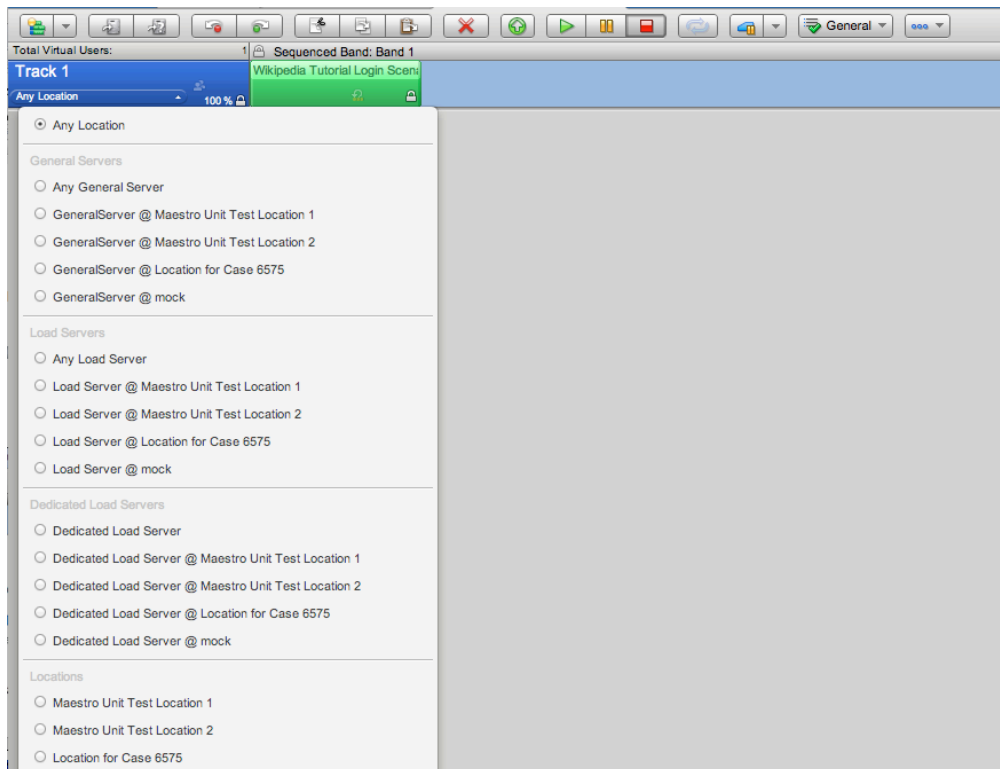
Specifying Locations using Composition Editor (Cloud Only)

Maestro is a test engine, implemented as a massively multi-threaded service, which executes all or parts of a complex test composition in SOASTA CloudTest.

A single Maestro node is capable of sending and validating responses to thousands of HTTP messages per second. Multiple Maestros can be combined to each execute parts of a large test that can scale out to tens and hundreds of thousands of virtual users. Maestros can be geographically distributed and single test compositions can run geographically distributed while still producing a single integrated set of test results and analytics.

You can specify a Maestro location for a specific track in one of several ways:

1. Click the down arrow in the select control underneath the track name in the left portion of the timeline. Then click the Maestro server or location that you want to play that track. Please note that if you have only one Maestro server defined in your SOASTA Repository, this drop down will not be visible.
 - Choosing a particular location means that you want to allow any Maestro at that location to control the track.
 - Choosing “Any Location” means that you do not wish to specify a preference as to which Maestro will control the playback of the track.



Maestro servers can also be selected from the lower panel of the Composition Editor by selecting a given track in the Property Explorer.

Using Load Composition to Prepare for Play

For complex tests using multiple server locations, it is useful to load the composition before pressing Play. To do so, click the Load Composition button (leftmost button below) on the Composition Editor toolbar.

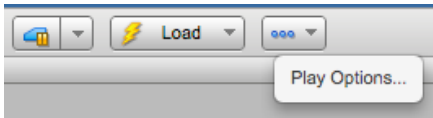


If the servers fail to start, examine your settings for errors using the Status message and the Details links on the upper right of the Composition Editor.

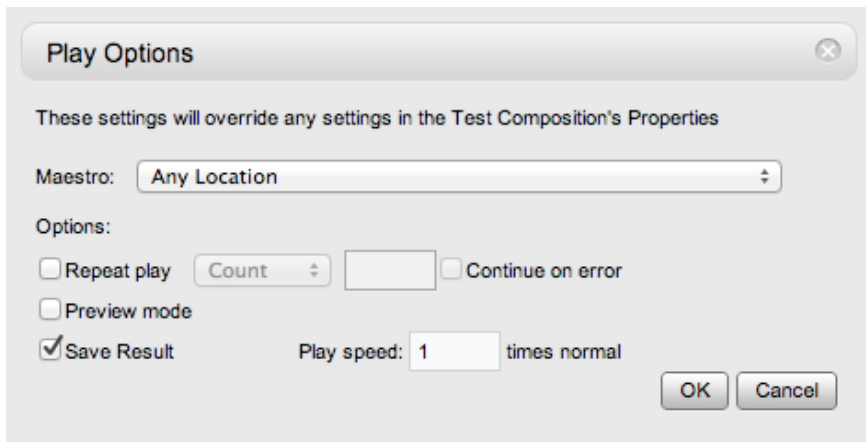
Using Play Options with Locations and Preview Mode

You can change the way a test composition will play by accessing Maestro locations and servers using the “Play Options” button in the Composition Editor toolbar. Changes will be in effect for only the current load of this test composition, not for the life of the test composition.

1. Click the Etc. (...) drop-down menu, and then select Play Options.



The Play Options dialog box appears.



2. Select a location from the Maestro drop-down.
3. Check Preview Mode if you'd like to stage server instances and run through the test without any messages being sent. Preview mode is useful in staging large tests that may involve multiple resources.
4. Click OK.

Playing the Test

1. When ready, click the Play button in the Composition Editor > Edit or > Play tabs.

Note: You can stop a running a test composition in case you've decided to abort it, however, the test composition may not stop immediately (stop represents a "request" not an "abort"). Closing the Composition Editor will not stop a running test composition.



The Composition Editor will indicate the test status on the upper right. Once the test begins to play, switch to the Play tab for runtime results. Up until now, all our work within Composition Editor has occurred in its Edit tab. SOASTA CloudTest presents test results within the Composition Editor > Play and > Results tabs.

1. While the test plays, the Composition Editor will switch to the Play tab.

The Play tab displays current results for the last or actively playing composition. The Play tab also provides buttons for creating dashboards, widgets, and to print the latest result for the test composition

2. When the test composition has finished playing, the Composition Editor switches to the Results tab. The Results tab shows all completed results with the most recent result displayed.

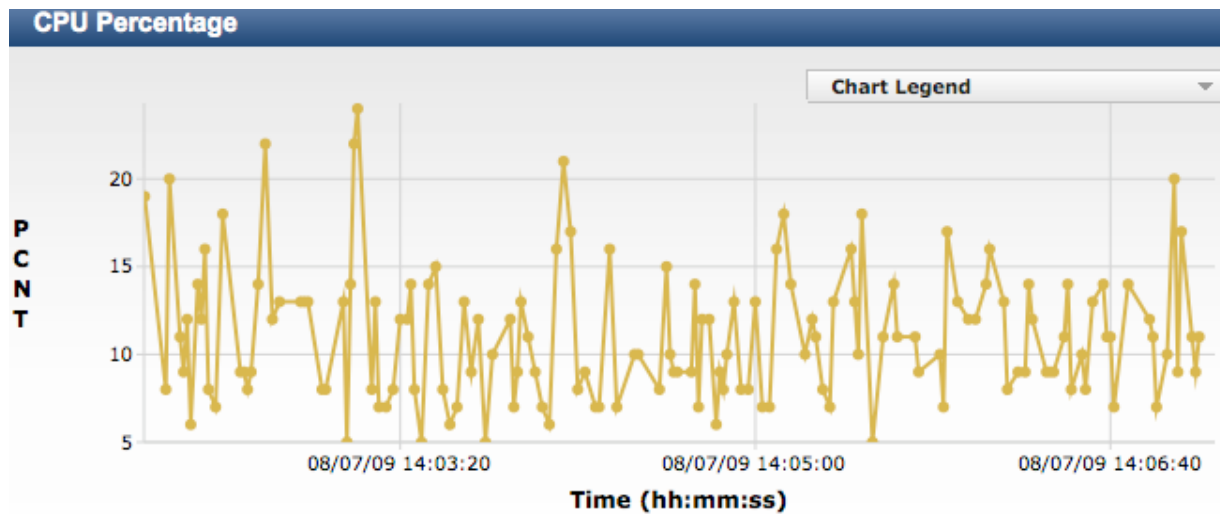
Analyzing Results

SOASTA CloudTest offers a comprehensive set of analytic widgets that you can place into one or more dashboards. When you click the Composition Editor > Results tab, the Analytic Dashboard appears with the Default Functional Dashboard attached by default and displaying results for this composition. You can duplicate any of the default dashboards using Central > Dashboards and then right-click, Duplicate.

See [Creating a New Dashboard](#) and [Adding a Widget to a Dashboard](#) for more information about configuring the dashboards and widgets associated with your test.

Adding a Monitor Widget to a Dashboard

Monitors can be added to dashboards in the same manner as other widgets. Monitors can also be viewed for a given time range by selecting Central > Monitors, and then clicking *View Analytics* to display a given Monitor in the list in its own window.



Debugging a Test

Debugging your test is an iterative process that identifies errors, and in the context of performance tests, performance bottlenecks by giving attention to the results.

Although, within the performance context we are not as interested in a Pass/Fail result or that Pass/Fail result with respect to the load thrown at the application(s) being tested, it may still be necessary to do some debugging to make those measures as useful as possible.

Note: Use Failure Action settings such as “Never halt on error...” in the Composition Editor > General properties to ensure tests run to completion without the failure that prevents performance measuring for the overall test.

Usually, debugging will begin with the Result Details widget, which helps you to identify when and where errors occurred. Result Details helps you determine the cause of errors in your test by browsing, as well as filtering for errors only.

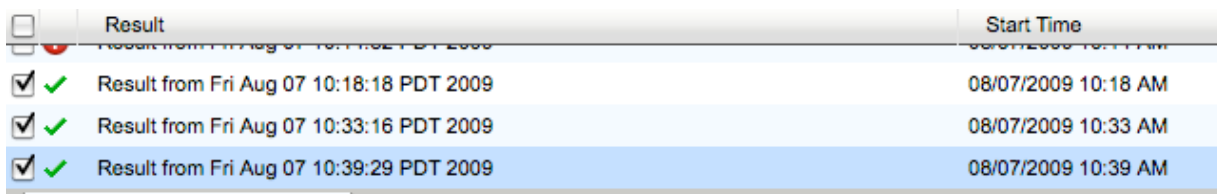
Tuning Performance Tests

Once you have established a low-end performance benchmark against your own site or application, ramp up the virtual users and then compare the subsequent test results to this benchmark.

Comparing Multiple Results

In the final section, we will demonstrate how to compare results from more than one test.

1. In SOASTA Central, click Central > Results.
2. Select two or more test results for your performance test.



<input type="checkbox"/>	Result	Start Time
<input checked="" type="checkbox"/>	Result from Fri Aug 07 10:18:18 PDT 2009	08/07/2009 10:18 AM
<input checked="" type="checkbox"/>	Result from Fri Aug 07 10:33:16 PDT 2009	08/07/2009 10:33 AM
<input checked="" type="checkbox"/>	Result from Fri Aug 07 10:39:29 PDT 2009	08/07/2009 10:39 AM

3. Right-click (or Control+click) to access the context menu and then select Compare in Dashboard.
4. When you do so, the Select Dashboard list opens. Select Result Comparison Dashboard from the list.
5. Mouse over the charts to get comparison information for the basic multi-results.



6. Toggle the lower-left property pane button to view a list of Multi-Result Dashboard charts on the left.


- At 100 versus 1000 users, the performance differences will usually be slight. To run larger tests against your own site, simply increase the number of virtual users. If you're running a test in the cloud, you can also define locations per track (Locations are configured by your SOASTA administrator).
- The Maximum Response Time by Result did vary far more widely. For a benchmark test with 100 users, we received 292 ms, while for a 1000 user test it was 3068 ms.

Refining Your Test

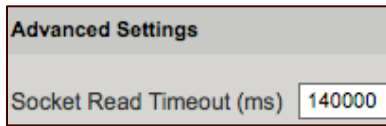
Initial runs of your performance test may include errors (depending on what you're testing) that indicate a need to refine that test. In these cases, your internal process may dictate your next move.

In the event that you'd like to further refine the test itself—we have the following suggestions.

- Time delays  can be added to increase the time before a message is fired. This can be useful when one or more test clip elements must succeed for the test itself to pass or where additional loading time is a given for the user.
 1. Select the insertion point, then click the Add menu, and then select Add a Time Delay  from the menu to insert the delay.

The new time delay will create some additional time between the previous message and the selected message.
- Think times (for sequential or manually-timed test clips) and bursts (for manually-timed test clips only) are both relevant to refining a performance test since they can either increase gaps in the test (think times) or ratchet the intensity (bursts) of message sends. This should be done as part of converting a recording to a test clip (refer to the Creating a Test Clip section at the beginning of this tutorial).
- Default timeouts can be set at the message level using the Clip Editor > Properties tab. To do so, open a message in the lower panel Message Editor by double-clicking it.
- Validations can be set to halt the test, either at the test composition level, or by use of validations at the test clip or message level.
- Checkpoints can be used to force a test to wait for a response. To do so, select the message or container that contains the message whose response must be received, and then click the Add menu, and select Add a Checkpoint  from the drop-down menu. This checkpoint will prevent the test from proceeding until the response is received.
- Socket Read Timeout can be used to manually force a longer timeout for a given message(s). To do so double-click a message to open its properties in the lower pane. Click the message name in the list below, and then scroll down until you see Advanced Settings in Element Info on the right. You can change the default

timeout for this message in order to prevent a timeout that is creating an error. The default value is 120000 ms.



- Repeat the above steps for all the test clip elements that are creating errors.
- Save your changes.
- Return to the Composition Editor and run the test again until you reach zero errors.

SOASTA, Inc.
444 Castro St.
Mountain View, CA 94041
866.344.8766
<http://www.soasta.com>