# Preparing Web Functional Tests for Load Testing

SOAtest web scenarios (including automatically-generated tests added when you record from a browser as well as manually-added Browser Playback Tool tests) are designed to be run in a browser. Since load tests don't run in a browser, some configuration is necessary to reuse functional web tests in a load testing environment—where web tests are conducted by sending requests to the server.

Parasoft SOAtest automatically configures your browser-based functional tests for load testing. It also validates them by executing them in a simulated load testing environment. This significantly reduces the setup required to create meaningful web load tests, and helps you to identify and resolve any potential load test issues before the load testing efforts actually begin.

This topic explains how to prepare your web scenarios for load testing. Sections include:



## Recommended Preparation Procedure

The recommended procedure is to configure your tests for load testing as described in Configuring Tests, then validate that they will work properly as described in Validating Tests.

However, if you do not want to run the configuration step (e.g., because you have already configured the tests and do not want to overwrite any manual configurations you added), configuration is not required as long as the validation step passes.

## Accessing and Understanding the Load Test Perspective

The Load Test perspective is designed to help you prepare your web scenarios for load testing. To open the Load Test perspective:

- Choose **Window> Perspective> Open Perspective> Parasoft Load Test**.

This perspective is similar to the SOAtest perspective, but it also provides the following features:

- Two toolbar buttons (Configure for Load Test and Validate for Load Test) which allow you to run automated test configuration and validation, as well as a toolbar button that launches Load Test.
- A Load Test Explorer, which lists the available web scenarios. Note that any scenario components that are not relevant to load testing—for example, browser-based validations or data banks—will not be shown in this view.
- Load Test Explorer right-click menus for running automated test configuration and validation (the same commands available in the toolbar buttons).
- Specialized test configuration panels, which are accessed by double-clicking a test in the Load Test Explorer.

## Configuring Tests

### Why Do I Need to Configure Tests?

Load tests take the set of requests that the browser test would use and sends those results outside of the browser context. Sometimes, browser requests become invalid when they are re-submitted outside of the browser—for instance, because a request contains session-dependent information such as session ID. In these cases, configuration is required. To facilitate configuration, SOAtest identifies such issues and automatically configures the requests to run in the browser-less load test environment.

In the configure mode, SOAtest:

1. Runs the test twice to identify dynamic parameters (e.g., session IDs).
2. Sets up a Text Data Bank to extract a valid value for each dynamic request parameter (e.g., using a value extracted from a previous test, or an earlier response in the current test). For more details about this tool, see Text Data Bank.
3. Configures the test to use the appropriate extracted value for each dynamic parameter. These requests are saved with the appropriate tests, and can be accessed as described in How Can I Review and Modify the Requests that SOAtest Configured? below.

This configuration is required when either:

- Load test validation does not succeed.
- Your application has evolved to the point that your existing load test configurations no longer match the application functionality.

### How Do I Configure Tests?

> **Warning**
>
> Configuration will re-create all the existing load testing requests based on the application's existing state. As a result, any existing load test configurations you have set up in SOAtest (for example, if you manually configured the URL or request body to be set using parameterized or scripted values) will be overwritten.

Run the automated configuration as follows:

1. In the Load Test Explorer, select the test suite that you want to configure.
2. Either click the **Configure for Load Test** toolbar button, or right-click the test suite and choose **Configure for Load Test** from the shortcut menu.

Next, validate tests as described in Validating Tests.

## How Can I Review and Modify the Requests that SOAtest Configured?

If you double-click on a Browser Playback tool in the Load Test Explorer (available in the Load Test perspective), you will see a special configuration panel that displays a list of the requests that the test is supposed to make. It shows both the URL, HTTP method, and request body, and allows you to modify these if desired. You can also add and remove requests using the controls on the right of the configuration panel.

Note that the method to invoke can be specified as a fixed value, parameterized value, or scripted value.

- For details about parameterizing values, see Parameterizing Tests with Data Sources, Variables, or Values from Other Tests.
- For details about scripting values, see Extensibility and Scripting Basics.
- With fixed values, you can access data source values using `${var_name}` syntax. You can also use the environment variables that you have specified. For details about environments, see Configuring Testing in Different Environments

## How Do I Parameterize or Script Request Values?

If you want to use a dynamic value for any part of the request, you can parameterize requests with values from a data source or values extracted from another test—or with values resulting from custom scripting or fixed values.

To do this:

1. Double-click the test in the Load Test Explorer (available in the Load Test perspective) to open its configuration panel.
2. Select the specific request whose values you want to parameterize.
3. In the **URL** or **Request Body** tab (depending on what part of the request you wish to parameterize), highlight the text you want to parameterize.
4. Click **Parameterize Selected Text**.
5. In the dialog that opens, specify a name for the parameterized value. The actual value in the URL or Request Body tab will be replaced with a reference to a variable, and an entry for that variable will be added to the Parameterized Values area at the bottom of the test configuration panel.
6. To configure the variable to use a fixed value, choose **Fixed** in the **Value** field, then select the desired value in the box to the right of **Fixed**.
7. To configure the variable to use a value that is stored in a data source or that is extracted from another test, choose **Parameterized** in the **Value** field, then select the desired data source column in the box to the right of **Parameterize**. See Parameterizing Tests with Data Sources, Variables, or Values from Other Tests for more details about parameterizing tests.
8. To configure the variable to use the result of a custom script, select **Scripted** in the **Value** field, then click **Edit Script** and specify the script details. See Extensibility and Scripting Basics for more details about using custom scripts.
9. If you want URL-encoding to be applied to the parameterized text before it is inserted into the larger value (either URL or Request Body) to which it belongs, ensure that the **URL-encode value** option is enabled.

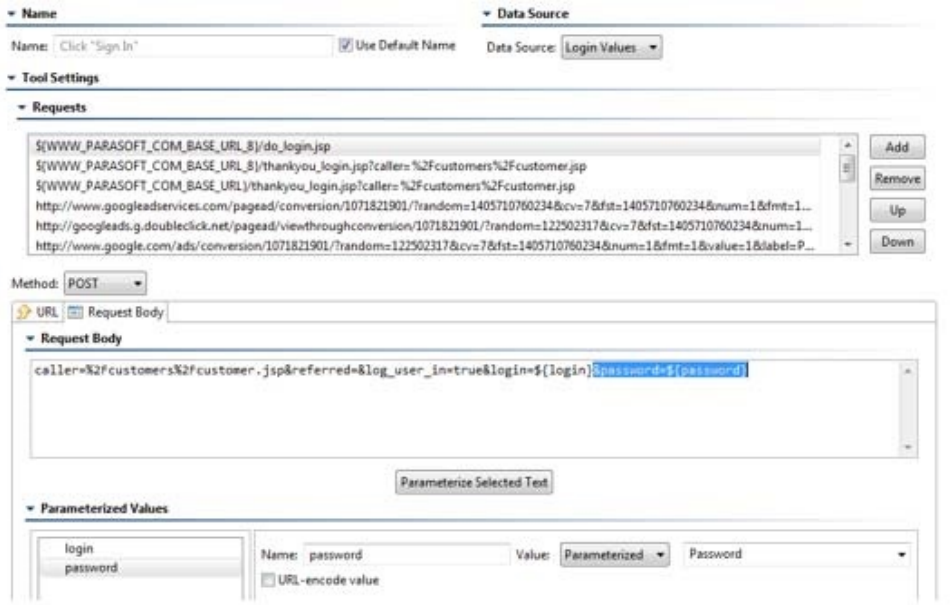
    - When "Configure for Load Test" is run and SOAtest automatically parameterizes dynamic values or values from a data source, the value of this option is set appropriately based on context.
        - **If the parameterized value belongs to the URL**, then this option will be auto-matically enabled after "Configure for Load Test" is run.
        - **If the parameterized value belongs to the Request Body**, then this option typically will NOT be automatically enabled after "Configure for Load Test" is run. However, if the Content-Type for the request body is "application/x-www-form-urlencoded", then this option will be automatically enabled. If the Content-Type is "multipart/form-data", SOAtest will not automatically URL-encode the value because multipart request bodies do not need to have the characters URL-encoded.
    - For instance, assume you are manually parameterizing part of the request body. If you want to parameterize it with data source values that need to be URL-encoded when sent as part of the request body—and the request body is not using the multipart format—then you would want to manually enable this option. Otherwise, the value would be inserted verbatim into the larger value.

Note that if your functional test is already configured to use parameterized values, the configuration step will set up the tests so that the parameterized values will also be used for load testing.

## What Happens During Test Configuration?

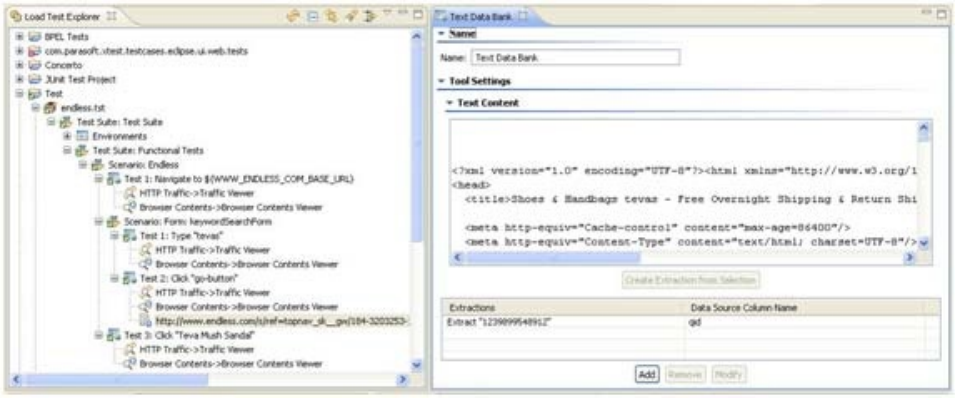
When running the "Configure for Load Test" step, SOAtest executes the test suite twice and performs three kinds of automated configuration:

1. **HTTP requests that were previously set to use data source values for the functional test scenario are automatically configured to use the same data source values for the load test configuration**
   Here is an example where SOAtest parameterized the username and password parameters in the HTTP request with the same data source values that the functional test was already configured to use:

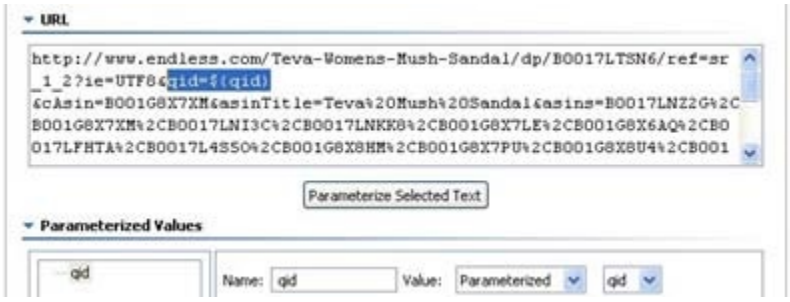



Note that the “password” parameter is configured to use the “Password” column from the "Login Values” data source.

2. **Dynamic parameter values (for example session ids) in the HTTP requests are configured to use the updated values from the current session**
   SOAtest does this by creating a Text Data Bank tool on an HTTP response that contains the dynamic value. This data bank value is then used in the appropriate HTTP request.
   For instance, in the example shown below, Test 2 had a Text Data Bank added to it.



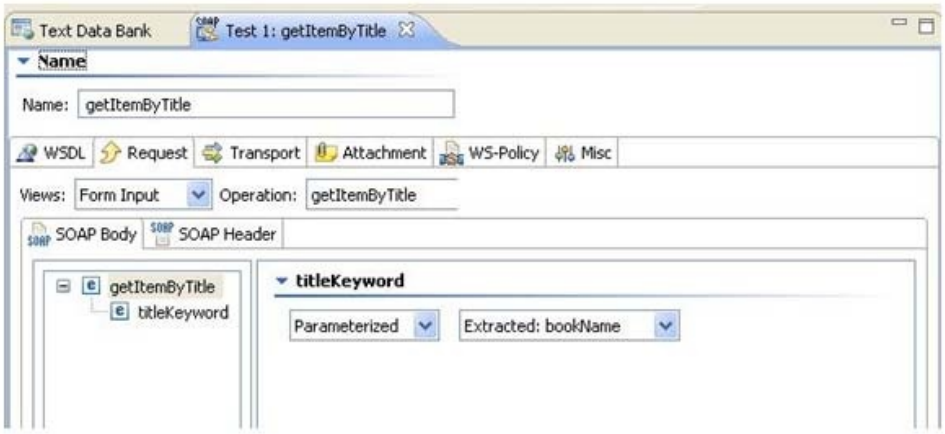

Note that a value is being extracted into a column name “qid”. Left and right hand text boundaries have been auto-configured.
One of the HTTP requests will be configured to use the extracted value:




For more details about this tool, see Text Data Bank.

3. **The scenario is configured to extract the same data bank values that were extracted for functional testing**
   This configuration makes the values available for other tools in mixed web/SOA load test scenarios.
   For example, assume that Test 2: Click “Google Search” has a Browser Data Bank that extracted a column named “Extracted: bookName”. SOAtest found the HTTP response that contained the same value, and created a Text Data Bank that extracts the value into a

column with the same name  (“Extracted: bookName”). This value is later used in a SOAP client, as shown below:




# Validating Tests

## Why Validate Tests?

When you validate tests, SOAtest will run them in load testing mode and alert you to any outstanding issues that might impact your load testing—for example, incorrectly configured HTTP requests. This way, you can resolve these issues before the actual load testing begins.

## How Do I Validate Tests?

Run the automated validation as follows:

1. In the Load Test Explorer, select the test suite that you want to validate.
2. Either click the **Validate for Load Test** toolbar button, or right-click the test suite and choose **Validate for Load Test** from the shortcut menu.

If the validation succeeds, the Validate for Load Test tab will report that the test suite is ready to be used in Load Test.

If a potential issue is detected, it will report that n issues need to be resolved before this test suite can be used in Load Test. You can then review the reported issues in the Quality Tasks view.

## How Can I See a Test Step Rendered in a Browser?

To better determine what is occurring at each test step, you can have SOAtest display what happens when the load test requests are rendered in a browser. To do this, double-click the Browser Contents Viewer added to the related test.

This is especially helpful if you want to visualize why the test is not producing the expected results. For example, the rendered page might reveal that the login is not occurring properly. Using this tool, along with examining error messages, helps you identify and resolve the cause of problems.

## What Is Validation Looking For?

During validation, SOAtest determines if any configuration needs to be done on the scenario—either automated configuration (by SOAtest) or manual configuration.  If validation does not succeed, this indicates that you need to run the configuration step or—if you have already run the configure step—that you need to manually configure parameters.

## What if Problems Are Reported?

If the dynamic parameters could not be auto-configured by “Configure for Load Test”, one or both of the following will happen:

1. Errors will be reported by “Validate for Load Test”.  Here are the kinds of errors you might see and what they could mean:
    a. HTTP error codes (e.g. 404 – Not Found or 401 – Not Authorized).  This means that the HTTP requests have incorrect dynamic parameter values or are otherwise wrongly configured.
    b. Functional test errors such as  “Unable to perform user action”, “Unable to validate or extract …”.  These errors occur because the specified page elements for the failing test could not be found.  Page elements not being found is typically the result of the HTTP responses containing unexpected data. Again, this is usually the result of the HTTP requests having incorrect dynamic parameter values or being otherwise wrongly configured.
2. The Browser Contents Viewer will show an incorrect or unexpected page at the point where the incorrect dynamic parameter was used.

If such issues occur, run “Configure for Load Test”.  If “Configure for Load Test” has already been run and these errors are still occurring, you may need to manually configure the HTTP requests and/or parameters causing the problem.

## When Do I Need to Manually Configure Parameters?

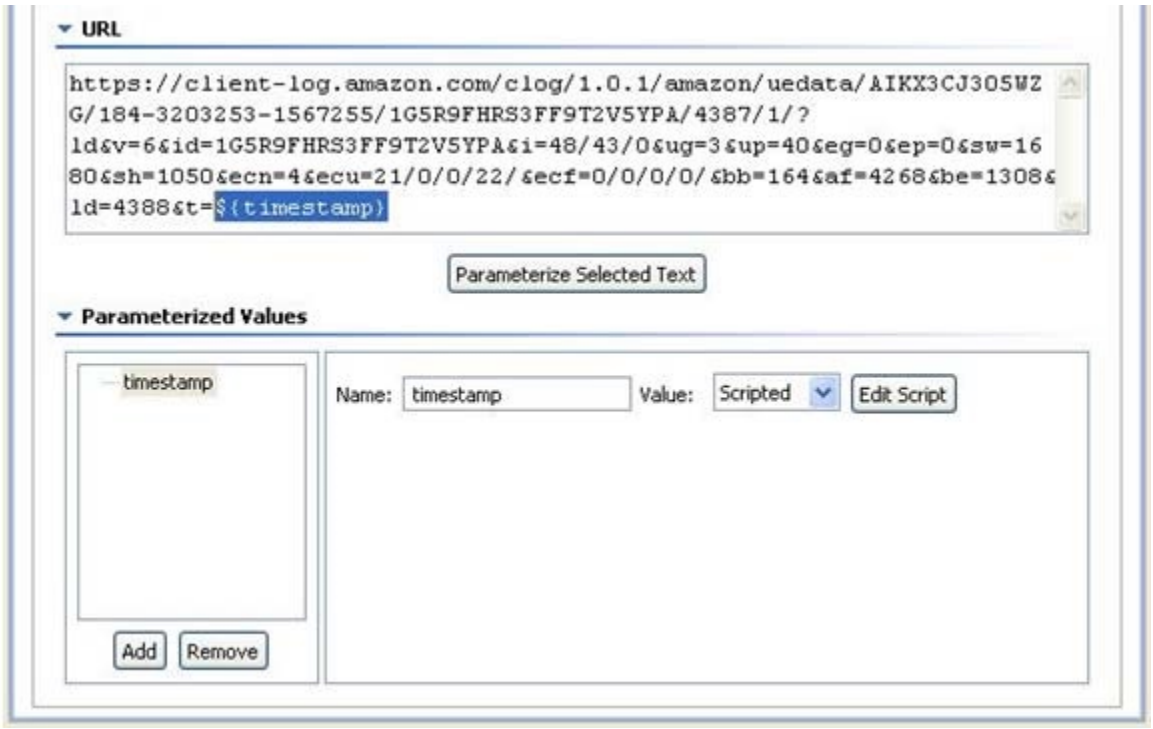
There is one class of dynamic parameter values that SOAtest cannot configure automatically: values that are normally constructed or transformed by JavaScript in the browser. Since the (transformed) parameter values do not exist in any of the HTTP responses, SOAtest cannot extract them to be used where necessary in any HTTP requests.

These parameters need to be configured manually. Validation will alert you when these kinds of dynamic parameters are present and are required by the web application to be updated for each session.

## How Do I Manually Configure Parameters?

Use the procedure described in How Do I Parameterize or Script Request Values.

Here is an example of a parameter that passes the current time to the server. This is a dynamic parameter, constructed by JavaScript, that is not present in any of the previous HTTP responses. It has been manually configured to be parameterized using a script that calculates and returns the current time.




# Web Load Testing Notes

- Web load testing focuses on requests that result in text responses. It does not transfer binary files such as images, flash files, JavaScript, CSS., etc.) This allows you to simulate a mode where everything is cached on the user's machine—providing response times that are accurate for a repeat visitor/existing user.
- The requests for web load testing are configured to simulate the browser specified in the test suite's **Browser Playback Options** tab. Browser type is simulated by sending the appropriate header content (User-Agent and Accept).
- If the application requires basic or NTLM authentication, the settings used in the test suite's **Browser Playback Options** tab will be applied to web load testing as well.

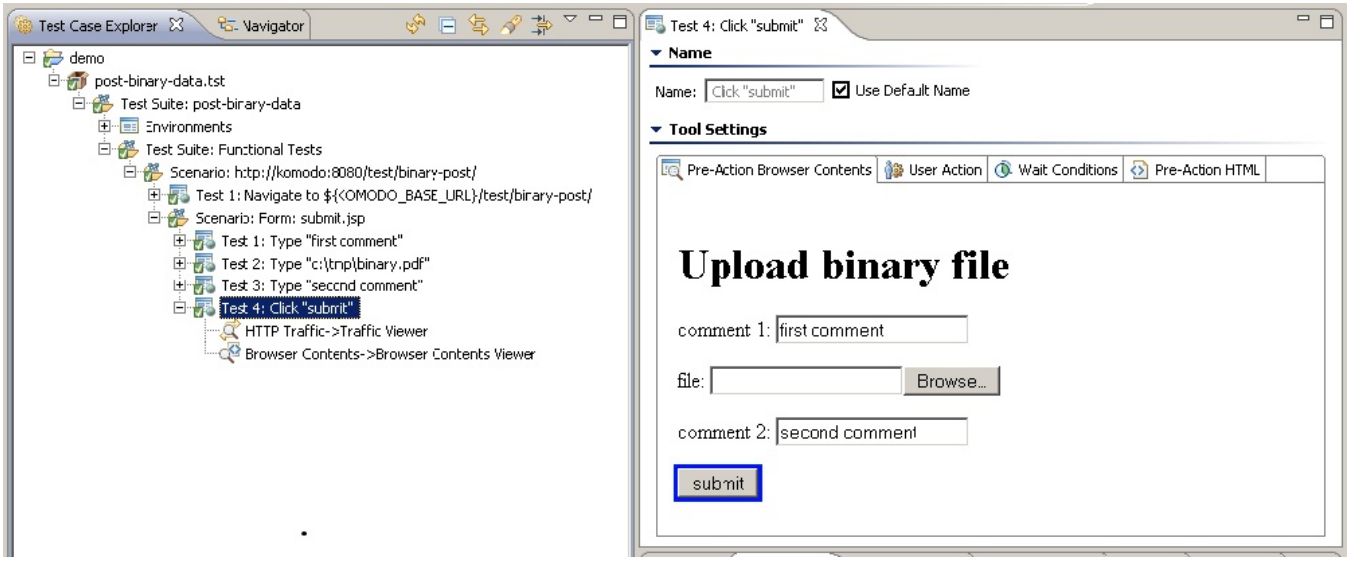
# Configuring Browser Tests to Send Binary Data

As mentioned above, the browser is not used when load testing a web scenario. Instead, Load Test sends a series of requests to simulate the result of a user action in the browser. Thus, if you want to send binary data in the browser request, you need to configure the test so that Load Test does not send the contents as text. The most common use case would be when you need to submit a binary file through a web page.

After recording the scenario, first run the scenario using the "Configure for Load Test" Test Configuration" (as described in Configuring Tests). This will configure the scenario with dynamic parameters needed to make the request during load testing.

For example, assume that you are uploading a PDF binary file. Normally, if you submit a file through a web page, the request will use a Content-Type of "multipart/form-data", where the request body will look something like this:

```
 ----------------------------7d9271373005fa
Content-Disposition: form-data; name="comment1"
first comment
----------------------------7d9271373005fa
Content-Disposition: form-data; name="binaryFile"; filename="binary.pdf" Content-Type: application/pdf
the contents of the file
which will contain binary data if this is a PDF file
 ----------------------------7d9271373005fa
Content-Disposition: form-data; name="comment2"
second comment
----------------------------7d9271373005fa
Content-Disposition: form-data; name="submit"
submit
----------------------------7d9271373005fa--
```

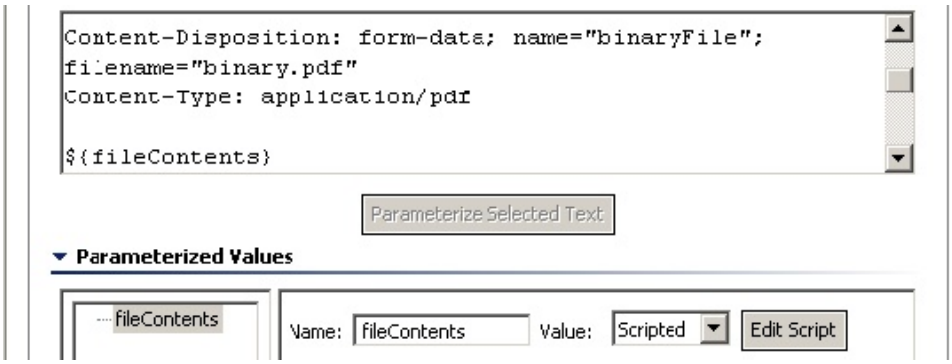



To configure this example:

1. In SOAtest's Load Test perspective, open the test that submits the PDF file. In the example scenario shown above, this is **Test 4: Click submit**.
2. In the test's **Request Body** tab, highlight the contents of the file and click **Parameterize Selected Text**.

> **What if the file contents don't fit in the Request Body text box?**
>
> If the PDF file is large, the contents might not fit in the text box that displays the request body. In that case:
>
> **a.** Set the scenario to use a small file.
> **b.** Run the "Configure for Load Test" Test Configuration.
> **c.** Create the parameterized value.
> **d.** Reset the scenario to used the desired PDF.

3. Name the parameterized value `fileContents` and use a **Scripted** value.




4. For the scripted value, use the following Jython code (substituting your own path to the binary file).

```
from com.parasoft.api import IOUtil
from java.io import File
def readBinaryFile(context):
     binaryFile = File("c:\\tmp\\binary.pdf")
     return IOUtil.readBinaryFile(binaryFile)
```

The scripted value must return a value of type byte[]. This is a Java type: you cannot return a Jython array. To create the byte[] value, you can use the utility method IOUtil.readBinaryFile(). You can also return a Jython jarray. (See the Jython documentation for more information on how to create a jarray object.)

Your request body should now use the parameterized ${fileContents} value:

```
-----------------------------7d9271373005fa
Content-Disposition: form-data; name="comment1"
first comment
-----------------------------7d9271373005fa
Content-Disposition: form-data; name="binaryFile"; filename="binary.pdf" Content-Type: application/pdf
${fileContents}
-----------------------------7d9271373005fa Content-Disposition: form-data; name="comment2"
second comment
-----------------------------7d9271373005fa Content-Disposition: form-data; name="submit"
submit
-----------------------------7d9271373005fa--
```

Now the contents of the PDF file will be sent as binary data when you either run the scenario in Load Test or when you run the scenario in SOAtest using the "Validate for Load Test" Test Configuration.

# Configuring Browser Tests to Add Custom Headers

Sometimes Validate for Load Test fails because the server requires an HTTP header to be present for any requests that it receives—but this header is actually absent. In such cases, you can add a global custom HTTP header to be sent along with requests during a web browser load test.

This is done by calling the BrowserTestUtil.addHttpHeader() method (which is documented in SOAtest's public API) in an Extension Tool within the context of the scenario in which you need to add the header. Be sure to position this Extension tool before the tests that you want to add that header. The header will be added to all scenario tests that are executed after the Extension tool is executed.The header will be added for any requests that these tools make during the load test.

If you have multiple scenarios, add this Extension tool into each scenario.  In such cases, we recommend defining this Extension Tool once as a global tool, and then adding it into the various scenarios where you want to use it. Or, you can create a .tst with it and access it as a reference .tst file.

Here is a sample Jython script that sets such a header:

```
from com.parasoft.api import *
def addHeader(x, context):
    BrowserTestUtil.addHttpHeader("HeaderName", "HeaderValue", context)
```

Note that this example uses Jython for the sake of simplicity, However, since this Extension tool will be used in the context of a load test, defining the script in Java will deliver the best performance:

## Supporting GWT Applications in a Web Load Test

In some cases, Google Web Toolkit applications append a custom HTTP header to all HTTP requests to prevent a security problem known as cross-site request forgery (XSRF). For example, this is done in applications using the smart-gwt toolkit. The custom header, which is called `X-GWT-Permutation`, is added to the request through GWT's JavaScript libraries.

In such situations, performing Validate for Load Test on GWT applications might cause a failure such as `500 - Internal Server` Error Looking at server-side logs, you might see the following error logged:

- `java.lang.SecurityException: Blocked request without GWT permutation header (XSRF attack?)`

This happens because JavaScript does not get executed in a load test context—and without JavaScript execution, the `X-GWT-Permutation` header does not get set as part of the requests.

To remedy this, write a short script to add this header so that it gets sent along with all requests during a load test. To do this, *add an Extension Tool before the first test in the scenario that fails because this header is absent.*

- **For the header name**: Use `X-GWT-Permutation`.
- **For the header value**: Use whatever value your application sets. To determine this, run the scenario as a functional test, and then look in the traffic viewers to determine the value for the header.