

Correlation in JMeter with an Example

What is correlation and why it is required?

Correlation is the most important aspect of scripting. It generally includes fetching dynamic data from preceding requests/calls and posting it to the subsequent requests.

Let's take an example to find out why exactly we need correlation

Suppose we have recorded a scenario in which

- The user enters login details and clicks the OK button
- The home page opens and the user takes further actions

Now, if we just playback this script, the test will fail even for a single user. This is because of the authentication mechanism used. When we log in to a website, session variables are dynamically created. These session variables are passed to the subsequent requests and help validation and authentication of the actions performed. So, one cannot just record and playback the requests having these variables. Here, we need to correlate the web requests with the dynamic variables. And for correlation, we need to use the "Regular Expression Extractor" which makes use of regular expressions.

A brief insight into regular expressions

Regular expressions are used to fetch data from a string based on a search pattern. Basically, what we do is in order to extract any value (generally a dynamically created value) from a string (text response), we define a left bound of the variable than some wildcard characters and then a right bound (Left Bound)(Wildcard Characters)(Right Bound)

For example: If we have a text response like;

```
.....__EVENTVALIDATION"value="weriudflsdfspdfusdjfsisdpfjpsdfohsdihffgdfgpdfjsdjfpj" />...
```

And we need to extract the value of the Event validation variable using regular expressions; the regular expression for the same will be

```
__EVENTVALIDATION" value="(.+?)" />
```

where, Left Bound = __EVENTVALIDATION" value="

Wildcard characters = (.+?)

Right Bound = " />

If you do not want to get deeper into regular expressions, then the wildcard characters (.+?) would suffice in most cases.

For more information on regular expressions and meaning of each wild card character visit;

<http://www.regular-expressions.info/tutorialcnt.html>

Regular Expression Extractor

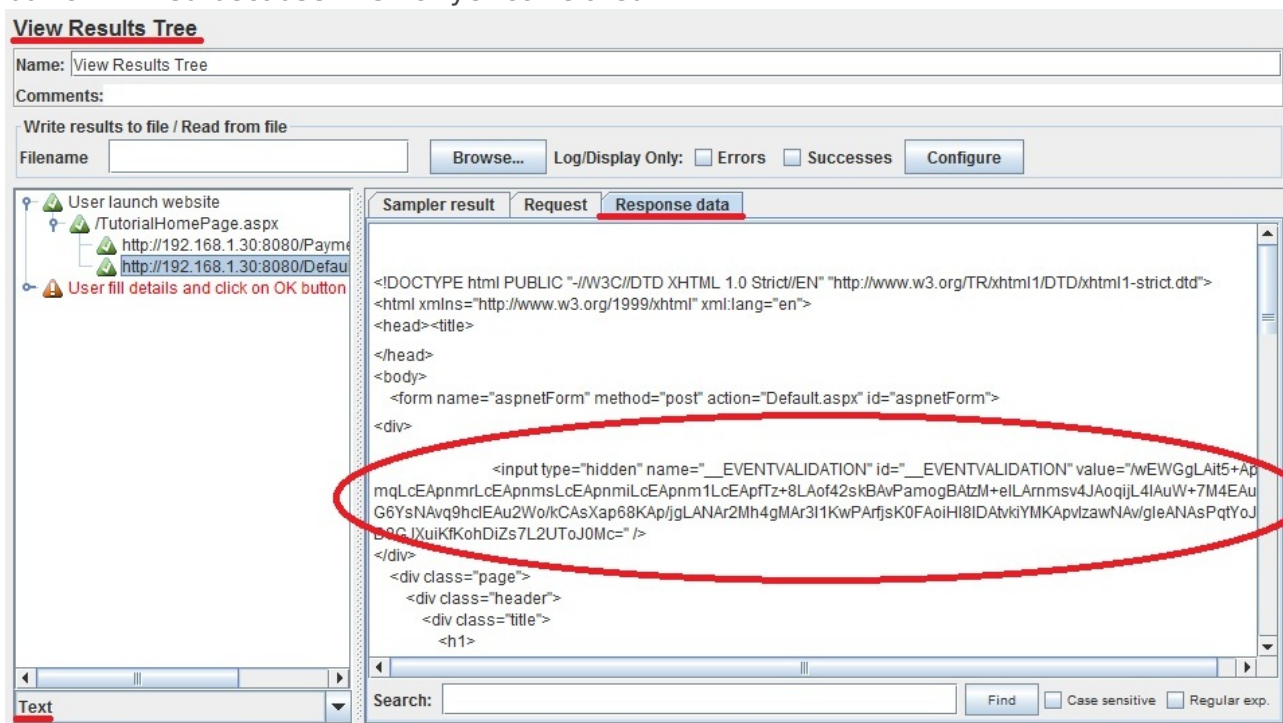
Coming back to JMeter, consider an example where we have two operations-

1. The user launch website
2. The user fills in details and clicks on the OK button

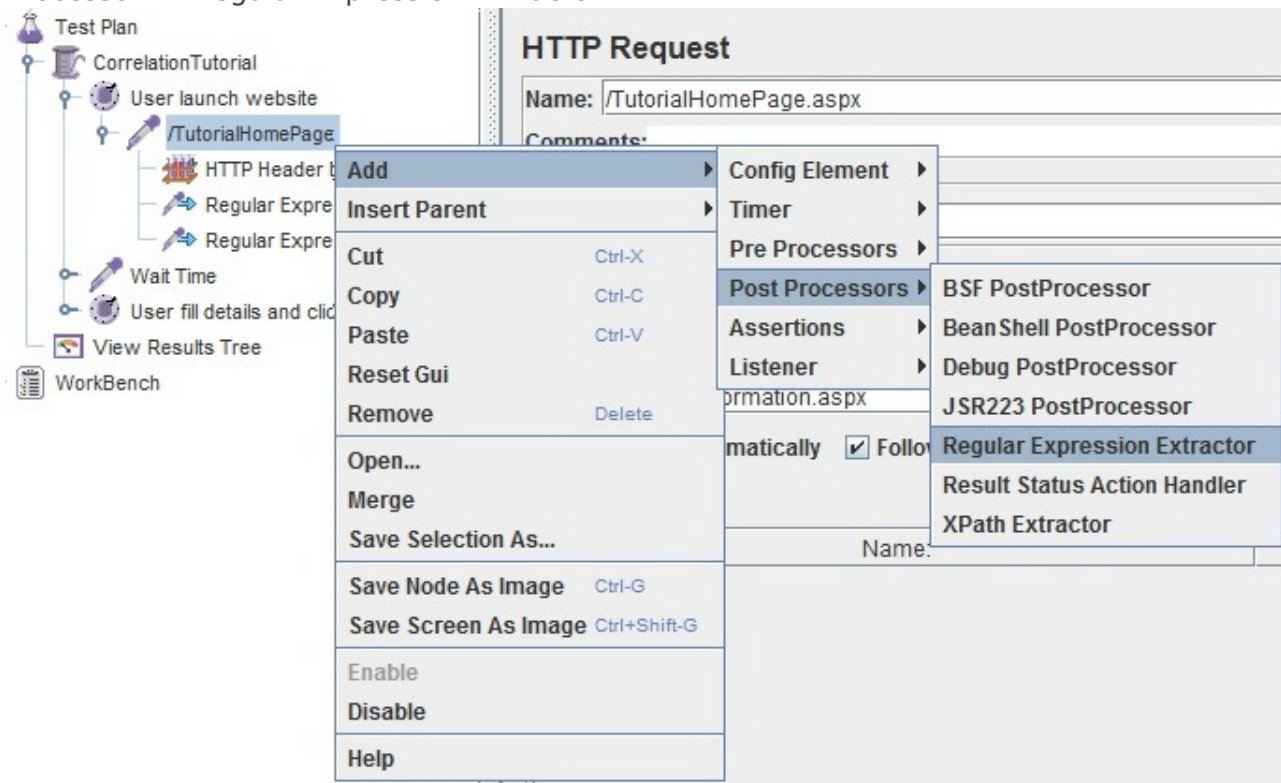
Now, the call user launch website creates a dynamic variable event validation that we can check in the Response Data tab of the "View Result Tree" listener for the call. The value of this variable is then passed to a subsequent call related to "User fill details and click OK button" as an HTTP post parameter.

Steps for correlating the Event validation values

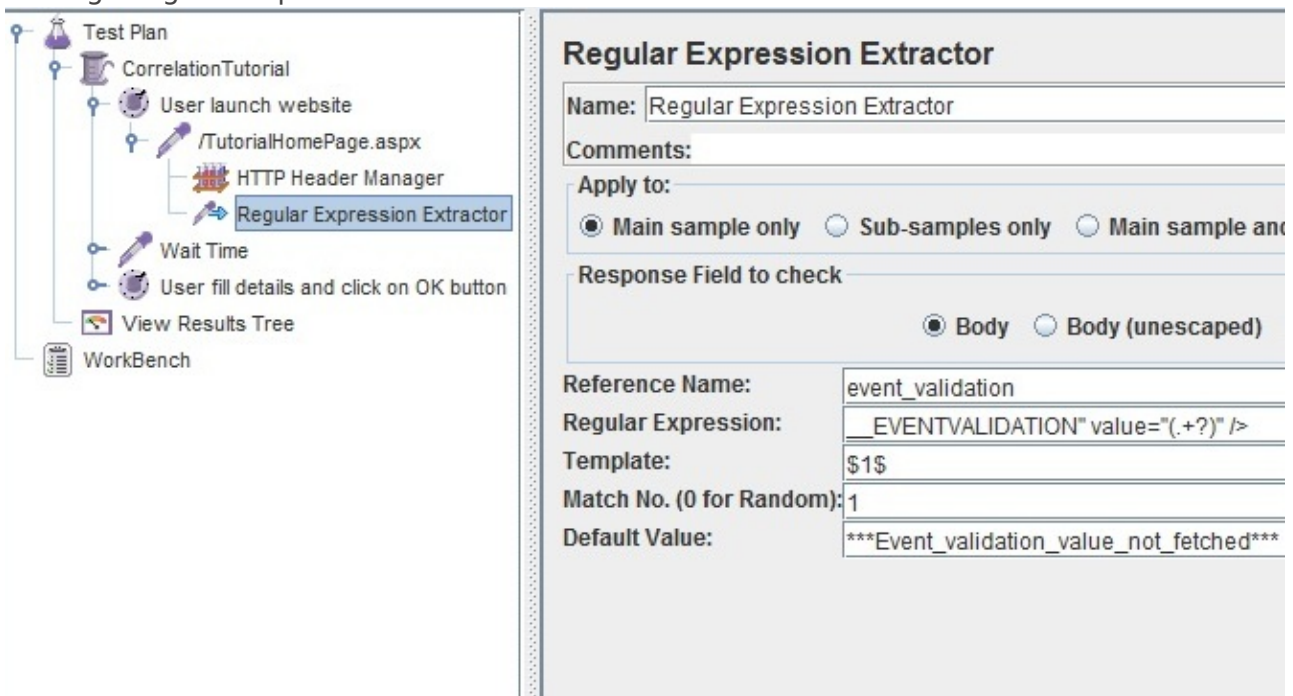
1. Run the script containing both the above-stated operations
2. Go to the Response tab (Text mode) in the "View Result Tree" listener of the "User launch website" operation. BTW, we see the second operation "User fill details and click OK button" in red because it is not yet correlated



3. Create a Regular expression for extracting the value of the Event validation variable's value. As stated above the R.E. for this will be-
`__EVENTVALIDATION" value="(.*?)"/>`
4. Go to HTTP request under "User Launch Website" transaction controller -> Add -> Post Processor -> Regular Expression Extractor.



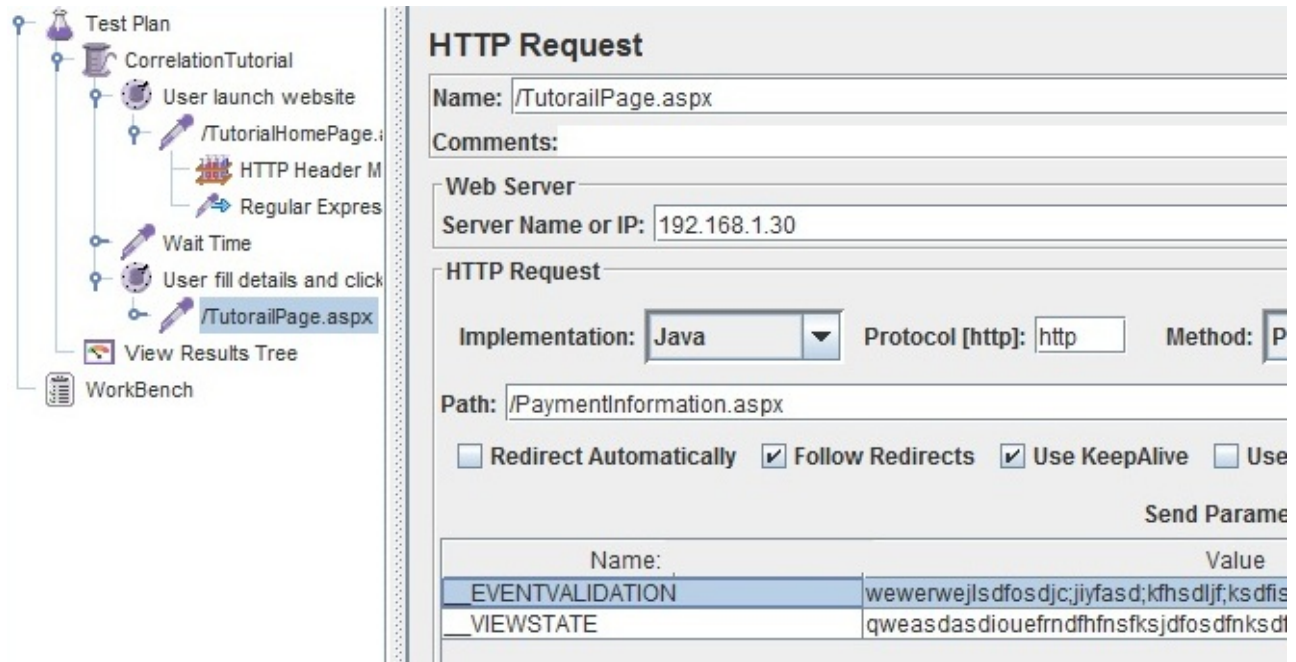
Adding "Regular Expression Extractor" control



R.E. Extractor Parameters Filled

5. The reference name inserted is the name of the variable created that will capture the Event validation value generated by the HTTP request under the "User launch website" operation
6. Now pass this variable to the subsequent HTTP request under "User fill details and click OK button" as a post request- overriding the already present hard-coded value of the

Event Validation variable.

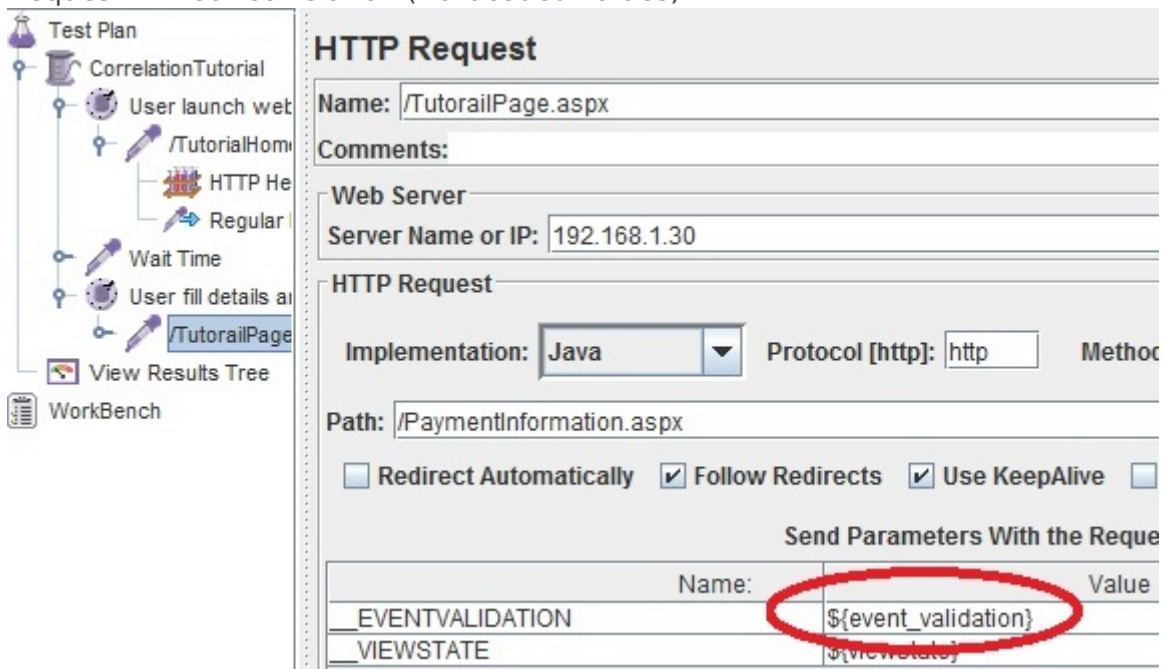


The screenshot shows the JMeter WorkBench interface. On the left, a Test Plan tree includes a 'CorrelationTutorial' folder with steps: 'User launch website', '/TutorialHomePage.aspx', 'HTTP Header Manager', 'Regular Expression', 'Wait Time', 'User fill details and click', and '/TutorialPage.aspx'. The 'HTTP Request' step is selected. On the right, the 'HTTP Request' configuration panel is shown with the following settings:

- Name: /TutorialPage.aspx
- Comments:
- Web Server:
 - Server Name or IP: 192.168.1.30
- HTTP Request:
 - Implementation: Java
 - Protocol [http]: http
 - Method: POST
 - Path: /PaymentInformation.aspx
 - ☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use
- Send Parameters With the Request:

Name:	Value
__EVENTVALIDATION	wewerwejl\$dfosdj;jiyfasd;kfh\$dljf;ksdfis
__VIEWSTATE	qweasdasdiouefndfhfnsfksjdfosdfnksdl

Request without correlation (Hardcoded values)



This screenshot shows the same JMeter configuration as above, but the values in the 'Send Parameters With the Request' table are dynamic variables, indicated by a red circle around the values:

Name:	Value
__EVENTVALIDATION	`\${event_validation}`
__VIEWSTATE	`\${viewstate}`

Request with correlation (Dynamic values)

7. Run the test plan again. All green? That's it.

This was all about correlation. In order to get a good understanding of correlation (or scripting for that matter), we need to have a good understanding of two things- the dynamic variables generated by the programming languages and Regular expressions.

SPREE APP

Let us take a working example of a Spree App where we will be recording the application and then correlating it

The URL for this Spree App is given below :

http://52.42.228.15:8888/

We will be using the above URL to record the below actions

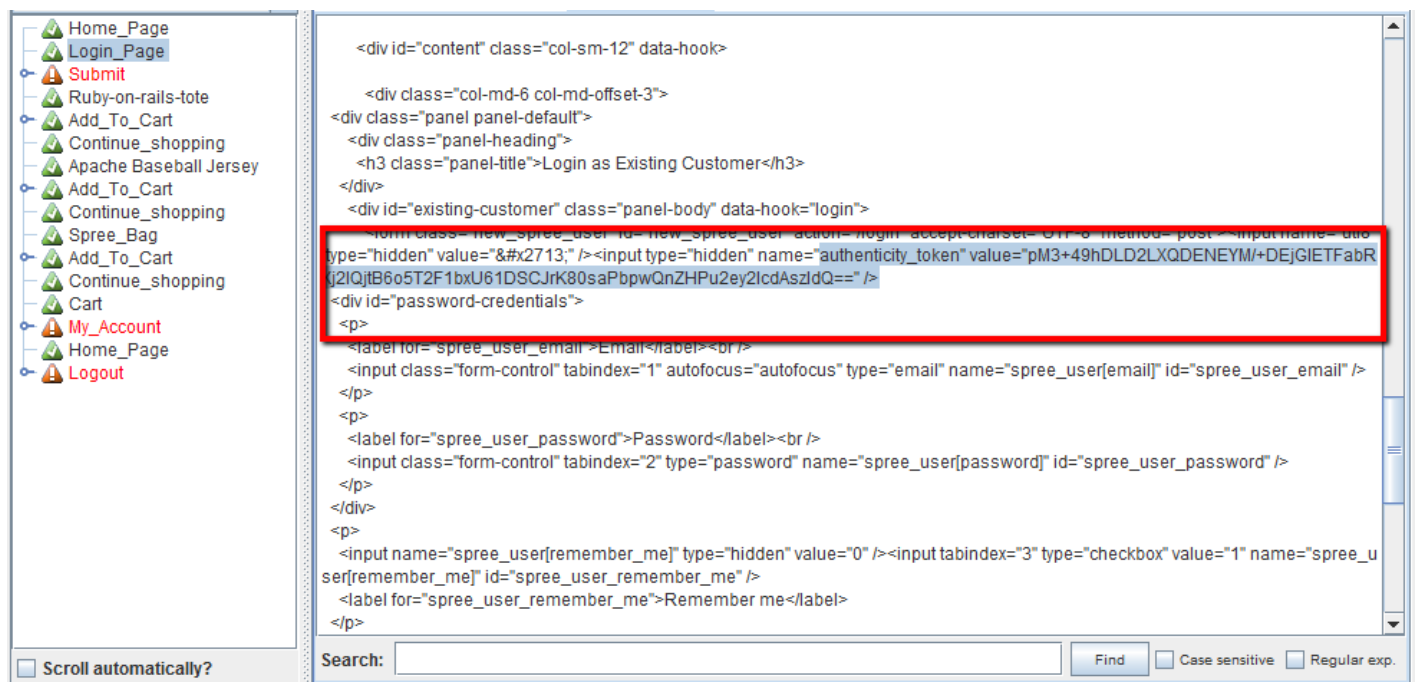
1. Navigating to the home page
2. Navigating to the Login page
3. Entering credentials and clicking on Submit
4. Clicking on ruby-on-rails-tote
5. Adding to Cart
6. Clicking on Products
7. Clicking on apache-baseball-jersey
8. Adding to Cart
9. Clicking on MyAccount
10. Navigating to the Home page
11. Logging out

Actions from Step 5 to Step 7 may be repeated depending on the requirement of the user. Once the above action is recorded user runs the recorded and checks the Output as shown below.

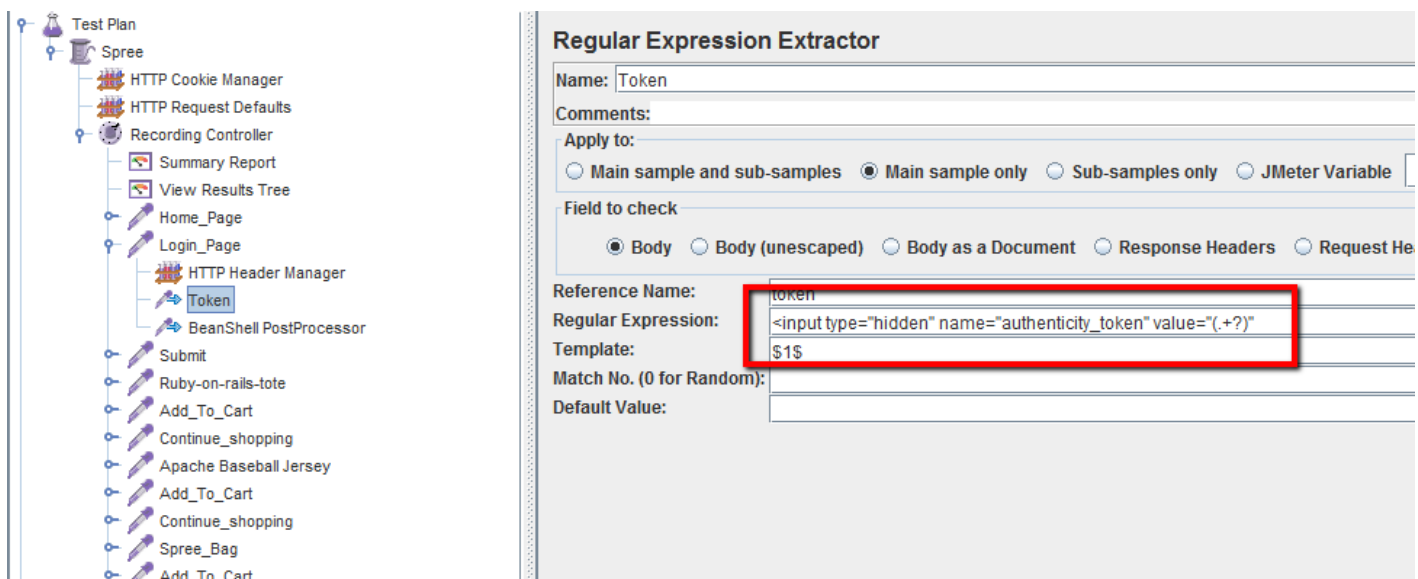
The screenshot displays the JMeter Test Plan on the left and the View Results Tree on the right. The Test Plan includes a 'Recording Controller' with a 'Summary Report' and 'View Results Tree' (highlighted). Below it, a list of test elements is shown: Home_Page, Login_Page, Submit, Ruby-on-rails-tote, Add_To_Cart, Continue_shopping, Apache Baseball Jersey, Add_To_Cart, Continue_shopping, Spree_Bag, Add_To_Cart, Continue_shopping, Cart, Empty_Cart, My_Account, Home_Page, and Logout. The View Results Tree on the right shows the execution results for these elements. It lists the same sequence of elements, with 'Submit', 'My_Account', and 'Logout' marked with red warning icons, indicating they failed. The other elements are marked with green checkmarks, indicating they passed.

As you can see from the above screen-shot there are a few requests like “Submit”, “My Account” and “Logout” that have failed.

By Checking the “Login Page” response data we find that there is an authentication token that is generated that is unique and should be used for all the failed transactions. The screenshot is given below.



As shown above we will have to use the Regular Expression extractor to extract the token and to be used in the Submit transaction.



The Reference name should be noted and the same should be used in the Submit transaction as shown below.

The screenshot displays a test plan editor interface. On the left, a tree view shows a sequence of steps: Test Plan, Spree, HTTP Cookie Manager, HTTP Request Defaults, Recording Controller, Summary Report, View Results Tree, Home_Page, Login_Page, HTTP Header Manager, Token, BeanShell PostProcessor, Submit, Ruby-on-rails-tote, Add_To_Cart, Continue_shopping, Apache Baseball Jersey, Add_To_Cart, Continue_shopping, Spree_Bag, Add_To_Cart, Continue_shopping, Cart, Empty_Cart, My_Account, Home_Page, Logout, WorkBench, and HTTP(S) Test Script Recorder.

The main panel shows the configuration for the 'Submit' HTTP Request. The 'Name' is 'Submit'. The 'Comments' state 'Detected the start of a redirect chain'. The 'Web Server' section shows 'Server Name or IP' as '52.42.228.15' and 'Port Number' as '8888'. The 'HTTP Request' section shows 'Implementation' as 'http', 'Method' as 'POST', and 'Path' as '/login'. Checkboxes for 'Redirect Automatically', 'Follow Redirects', 'Use KeepAlive', 'Use multipart/form-data for POST', and 'Browse' are present.

The 'Parameters' tab is active, showing a table of parameters to be sent with the request:

Name	Value
utf8	
authenticity_token	\$(token)
spree_user[email]	\$(user)

Buttons for 'Detail', 'Add', 'Add from Clipboard', 'Delete', and 'Up' are located below the table. The 'Send Files With the Request' section is empty, with buttons for 'Add', 'Browse...', and 'Delete'.

The 'Proxy Server' section is empty, with fields for 'Server Name or IP', 'Port Number', and 'Username'. The 'Embedded Resources from HTML Files' section is also empty.

Replace the same in the transaction which needs the Authenticity token. Save the script and run the test again.

Test Plan

Spree

HTTP Cookie Manager

HTTP Request Defaults

Recording Controller

Summary Report

View Results Tree

Home_Page

Login_Page

HTTP Header Manager

Token

BeanShell PostProcessor

Submit

Ruby-on-rails-tote

Add_To_Cart

Continue_shopping

Apache Baseball Jersey

Add_To_Cart

Continue_shopping

Spree_Bag

Add_To_Cart

Continue_shopping

Cart

Empty_Cart

My_Account

Home_Page

Logout

WorkBench

HTTP(S) Test Script Recorder

View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename

Text

Home_Page

Login_Page

Submit

Ruby-on-rails-tote

Add_To_Cart

Continue_shopping

Apache Baseball Jersey

Add_To_Cart

Continue_shopping

Spree_Bag

Add_To_Cart

Continue_shopping

Cart

My_Account

Home_Page

Logout

Sampler result

Request

Respo

Scroll automatically?

Search:

Since we have extracted the token and used it in the transactions wherever necessary all the steps would now pass.

Last updated: July 05, 2023

If you have any feedback on this topic, please contact us at documentation@appvance.ai