Performance Testing SAP BusinessObjects Enterprise XI3.1 and Web Intelligence with Apache JMeter

Applies to:

SAP BusinessObjects Enterprise XI 3.1 SP4 (steps relevant to all versions of XI 3.1). For more information, visit the Business Objects homepage.

Summary

This article provides directions for configuring Apache JMeter to execute a performance test with SAP BusinessObjects Enterprise XI 3.1 SP4 and Web Intelligence. It provides background on the JMeter components necessary to create a performance test and explains the fundamentals of extracting dynamic data from a complex web application.

Author: James Rapp

Company: SAP

Created on: 29 June 2011

Author Bio

James is a Senior Ecosystem Quality Manager at SAP working closely with OEM partners. James' experience with SAP BusinessObjects products stems from his previous experience as a Technical Account Manager for large enterprise deployments as well as an Onsite Technical Specialist in Business Objects Customer Assurance.

Table of Contents

Introduction to Apache JMeter

In this document, we will address the specific process for designing and creating a performance test in Apache JMeter. The basic fundamentals of the tool will be covered, along with detailed process flows for benchmarking Web Intelligence. Once the basic components of report execution and navigation have been unlocked, we will be able to create complex performance tests based on a variety of content types and workflows.

The exercise should result in a fully functional performance test using SAP BusinessObjects Enterprise XI 3.1 SP4 and the eFashion sample database that ships with the Windows version of the product.

Apache JMeter is open source software, a 100% pure Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

Where to Get JMeter – JMeter can be downloaded from the following location:

http://jakarta.apache.org/site/downloads/downloads_jmeter.cgi

How to Install JMeter

Installing JMeter is as simple as extracting the zip file to a desired location. Like many Jakarta programs, it is a self contained java application and does not require an installer.

How to Launch JMeter

JMeter can be launched by double-clicking the jmeter.bat file in the bin directory of the application.

System Resource Considerations

JMeter can be run in a 32-bit or 64-bit JVM, ensuring that the application can be allocated a significant amount of memory. Large tests can be both CPU and Memory intensive, depending on the complexity of the test, and the amount of data collected. When collecting statistics such as the listener 'View Results Tree', which collects the entire page content for each request, memory consumption grows at a much quicker rate. For example, in a 25 user test viewing and refreshing a Web Intelligence report, java.exe consumed over 500MB of memory, and generated java.lang.outofmemory exceptions with the default JVM parameters.

JMeter Memory can be adjusted by modifying the jmeter.bat file and changing the HEAP parameter. The example below demonstrates changing the heap to 1024 MB:

set HEAP=-Xms1024m –Xmx1024m

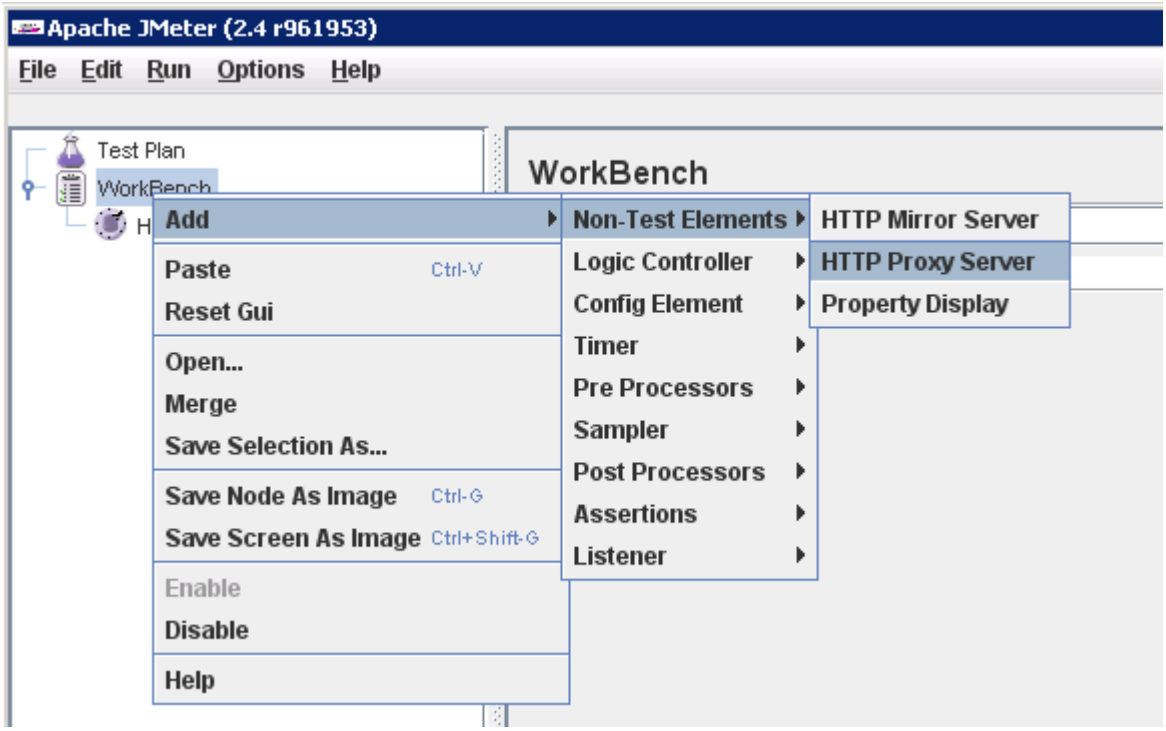
Save the file and restart JMeter.

Execution

The first step to actually recording a test plan in JMeter consists of recording the necessary workflows using an HTTP Proxy component. A performance test can be created manually as well, by generating the necessary URI requests by hand and assigning them to samplers, but for our purposes would represent too much manual effort to be productive. If the intent were, instead, to simply execute a large volume of Open Document requests, this might be a viable alternative.

After launching JMeter, perform the following steps to add an HTTP Proxy component to your performance test:

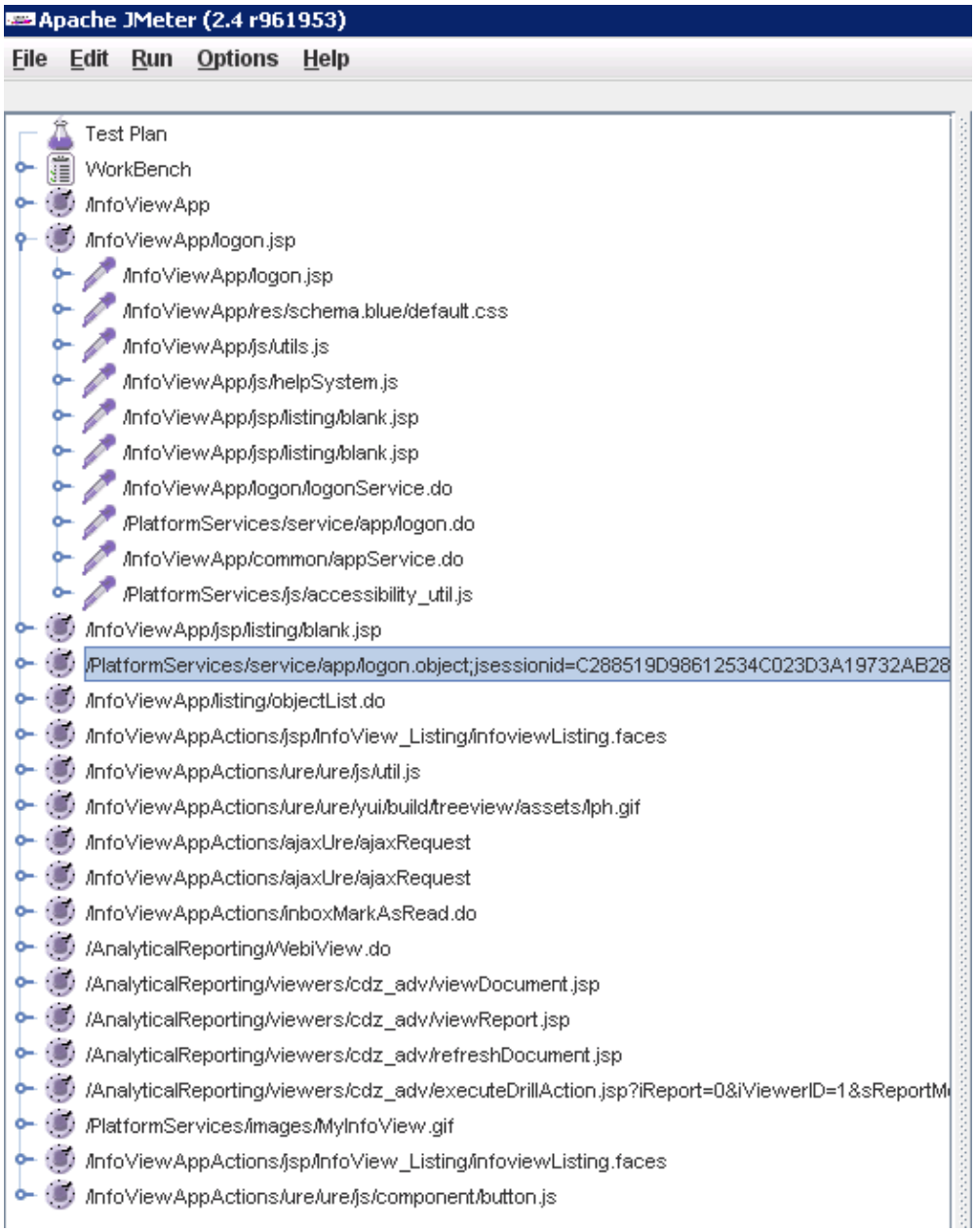
Capture HTTP Requests with HTTP Proxy



1. Right click Workbench > Add > Non-Test Elements > HTTP Proxy Server
2. Set the browser to use localhost:8080 as a proxy:
    - Launch Internet Explorer
    - Select Tools > Internet Options > Connections > Local Area Connection (LAN) Settings
    - Check the box, “Use a proxy server for your LAN”
    - Enter localhost in the address field
    - Enter 8080 in the port field
    - Uncheck the box, “Bypass proxy server for local addresses” and click Ok to exit all open menus
3. In JMeter, set the Grouping option to “Put each group in a new transaction controller ”
4. Check the box ‘Regex Matching’
5. Start JMeter HTTP Proxy Server
6. Clear browser cache to ensure all requests are made to the server
    - Select Tools > Internet Options > Delete Files (Under Temporary Internet Files or Browsing History depending on version)
    - For Internet Explorer 8, select Temporary Internet Files from the ‘Delete Browsing History’ menu that pops up upon selecting Delete.
7. Login to InfoView using http://servername/InfoViewApp/logon.jsp

8. Select Document List, Browse to the folder ‘Feature Samples’ and view the report ‘Formatted Report for Drill’
9. Select ‘Refresh Data’
10. Drill into the first available cell under ‘State’ (New York)
11. Close the report
12. Logoff
13. Stop JMeter HTTP Proxy

You should now have a recorded test scenario that looks something like the following:



The images above that look like timers are known as “Transaction Controllers”. The relevance of the transaction controller is that it generates an additional sample which measures the overall time taken to perform the nested test elements. This allows us to record response time and throughput for a series of related actions, instead of only measuring the response time for each page request, such as blank.jsp, which may be only a few milliseconds and not relevant to our analysis.

It is generally preferable to do some consolidation of the requests, as ultimately we want to measure the operations we identified in the previous step. That is:

- Login
- Navigate
- View/Refresh
- Drill
- Close Report/Logoff

As such, we can simplify our recorded steps by making a few modifications:

- Expand the controller for /PlatformServices/service/app/logon.object
    - Highlight all samplers under this controller (a sampler is represented by an eye dropper image) and select 'Cut' from the right-click menu
    - Highlight the /InfoViewApp/logon.jsp, right click, and select paste to add the samplers to this transaction controller
- Repeat this process for the remaining controllers
- Upon completion we should have the following top level transaction controllers:
    - /InfoViewApp/logon.jsp
    - /InfoViewApp/listing/objectList.do
    - /AnalyticalReporting/viewers/cdz_adv/viewDocument.jsp
    - /AnalyticalReporting/viewers/cdz_adv/refreshDocument.jsp
    - /PlatformServices/images/MyInfoView.gif
- This now leaves us with 5 independent transaction controllers that we can rename to more meaningful names.
- To rename, select the transaction controller /InfoViewApp/logon.jsp from the workbench, and change the 'Name:' field to ***Login to InfoView***
- Check the box 'Generate Parent Sample' in order to record statistics for the entire controller with each URI as a child object
- Repeat the process using the following names for the remaining controllers
    - ***Navigate Document List***
    - ***View Web Intelligence Document***
    - ***Refresh and Drill in Web Intelligence Document***
    - ***Close Report and Logoff***

Move recorded transactions to Test Plan

1. Right click 'Test Plan' > Add > Threads (Users) > Thread Group
2. Using the shift button, highlight all of the requests from Workbench and choose CUT
3. Right-click on 'Thread group' and choose PASTE (This moves all requests to the 'Thread group')
4. Select the Thread Group and change the name to 'Web Intelligence – Refresh and Drill'
5. Save the Test Plan by selecting File > Save Test Plan As > JMeter - Exercise 1 Webi

Add Supporting Test elements to the Test Plan

1. Add an HTTP Cookie Manager to allow JMeter to store unique cookies for each distinct user
    - Right click 'Test Plan' > Add > Config Element>HTTP Cookie Manager
    - Check the box 'Clear cookies for each iteration' so that new cookies are used each time we loop through the Thread Group (for large, iterative tests)
2. Add an HTTP Request Defaults element to configure some defaults for all threads to share
    i. Right click 'Test Plan' > Add > Config Element>HTTP Request Defaults
    ii. Enter the web server name (or IP address) in the 'Server Name or IP' field
3. Add a CSV Data Set to store dynamic username and password information for JMeter to leverage.

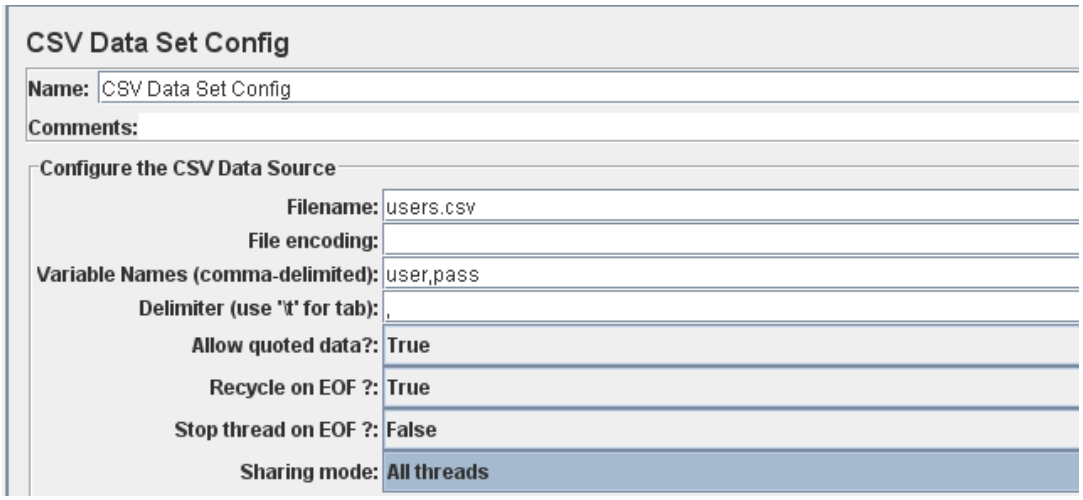
Note: This step assumes that you have already created the necessary users and groups, and granted them appropriate permissions to Web Intelligence (such as the ability to drill), within SAP BusinessObjects Enterprise XI3.1. There are tools that can be used for this such as:

BOE Stress Tester – This tool has a user creation property that is useful for creating a large number of users. It can be downloaded from:

http://boestresstester.sourceforge.net

Import Wizard – The Import Wizard is capable of creating users from a text file. Please refer to SAP Note 1408767 or Chapter 6 of the XI3.1 Import Wizard guide for more details. The Account Manager feature in the CMC can be used to set the password of each account to some value, such as 'password'.

- Right click 'Test Plan' > Add > Config Element>CSV Data Set Config
- Select the CSV Data Set Config object and change the name to 'Username and Password CSV Data Set Config'
- JMeter looks for the CSV file in the same directory that the project file is located by default. If the file is stored elsewhere, be sure to enter the full path to the file in the 'Filename' field. Create a file called users.csv in the above directory and add the usernames/passwords that you want to be used in the test. For example:
  `PerfTest001,password`
  `PerfTest002,password`
- Enter users.csv into the Filename field of the CSV Data Set Config element
- Enter the variable names to be used into the corresponding field, as below. These are the values that will be used to make the username and password entries dynamic for each thread.
- Enter a comma for the delimiter as below:



Configuring Dynamic Data in the Thread Group

Now that the core elements of the test plan have been created, we can begin to customize the scenario to ensure we are successfully able to simulate a large number of users with independent sessions and cache.

This process can be quite lengthy, so please remember to save your test plan frequently.

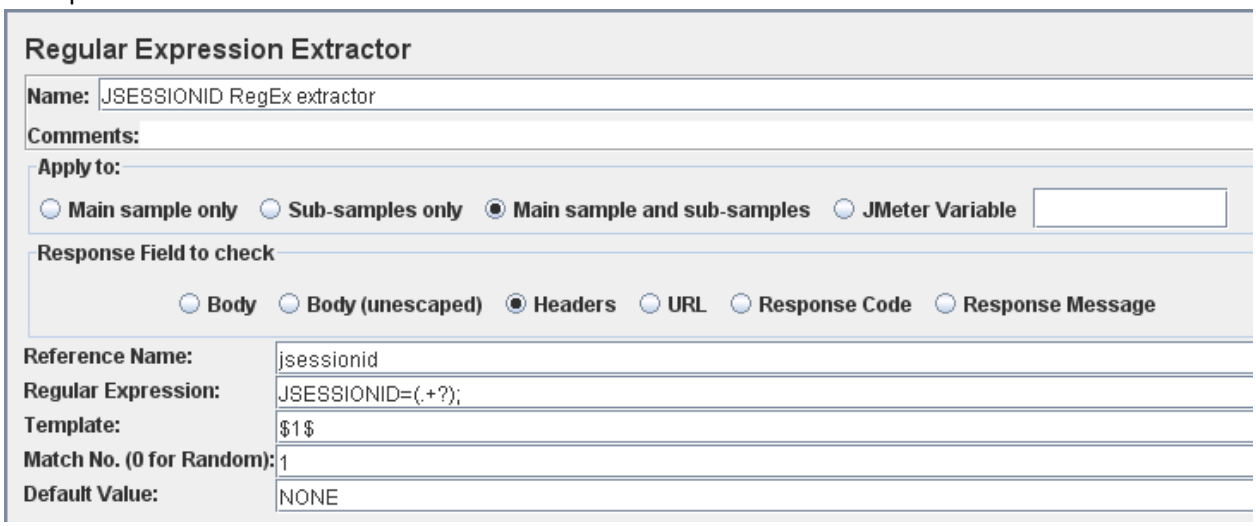
1. First, we attach a Regular Expression Extractor to the logon.do request in order to extract the uniquely generated JSESSIONID from the HTTP Response Header.

   Select /PlatformServices/service/app/logon.do under 'Web Intelligence – Refresh and Drill' > '***Login to InfoView***' and Right-Click> Add > Post Processor > Regular Expression Extractor.

   Configure it as follows:

   - Name: JSESSIONID RegEx extractor
   - Apply to: Main sample and sub-samples
   - Response Field to check: Headers
   - Reference Name: jsessionid
   - Regular Expression: JSESSIONID=(.+?);
   - Template: 1
   - Match No: 1
   - Default Value: NONE

Example:



This component looks at the HTTP header returned by logon.do and extracts all characters between JSESSION= and a semi-colon. This allows us to store the unique JSESSIONID for each user.

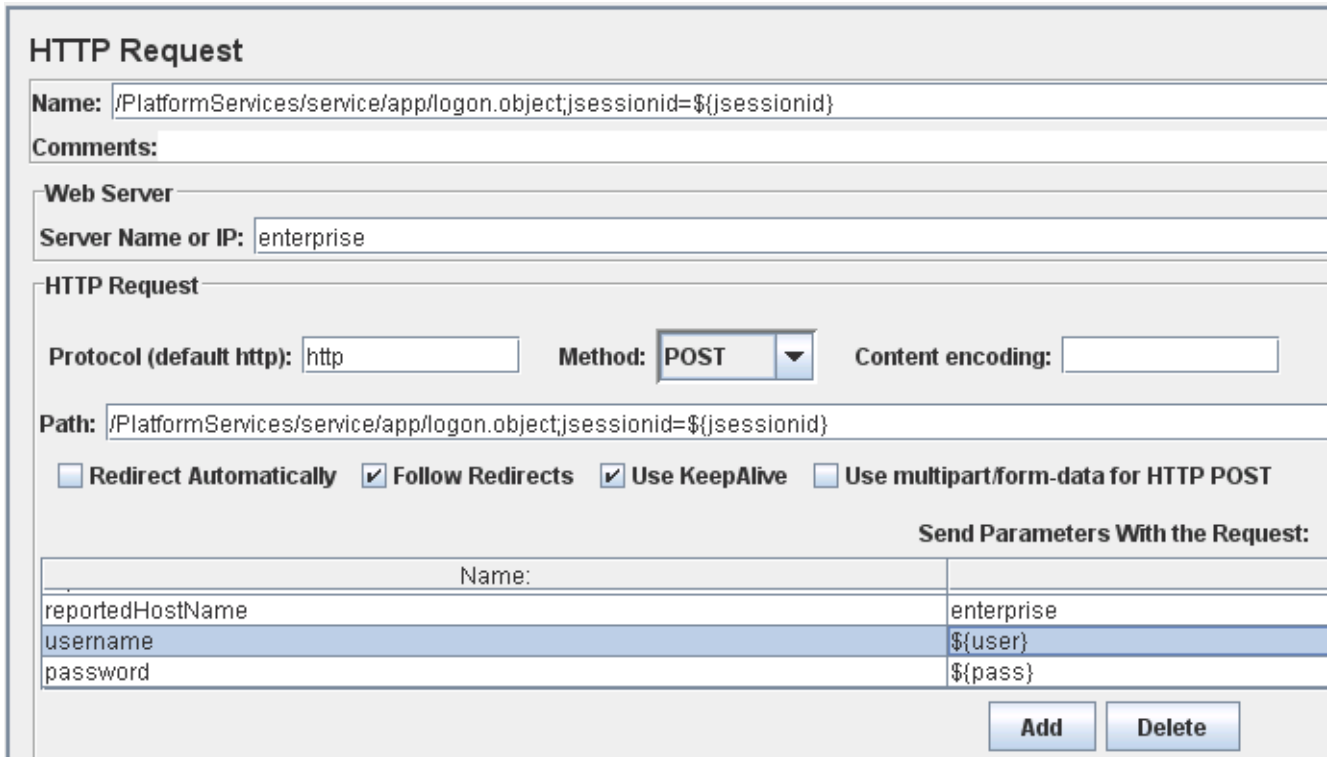
We utilize JMeter's Regular Expression extractor to evaluate the response body of a file (such as DHTML or AJAX) or HTTP Headers and extract values for use in later requests. We will use the Regular Expression extractor frequently when creating performance tests.

2. Next, apply the dynamic user data (username and session) to the logon object.

   Select '/PlatformServices/service/app/logon.object;jsessionid=' under 'Web Intelligence – Refresh and Drill' > '***Login to InfoView***' and make the following changes:

- Name:
  /PlatformServices/service/app/logon.object;jsessionid=${jsessionid}
    - Changing the name of the sampler does not have any functional impact on the test, but is useful as a reminder that this request contains dynamic data
- Path:
  /PlatformServices/service/app/logon.object;jsessionid=${jsessionid}
    - This modifies the request to logon.object so that it applies the contents of the jsessionid variable extracted above
- Username: ${user}
    - This applies the username variable defined in the 'Username and Password CSV Data Set Config'
- Password: ${pass}
    - This applies the password variable defined in the 'Username and Password CSV Data Set Config'

Example:



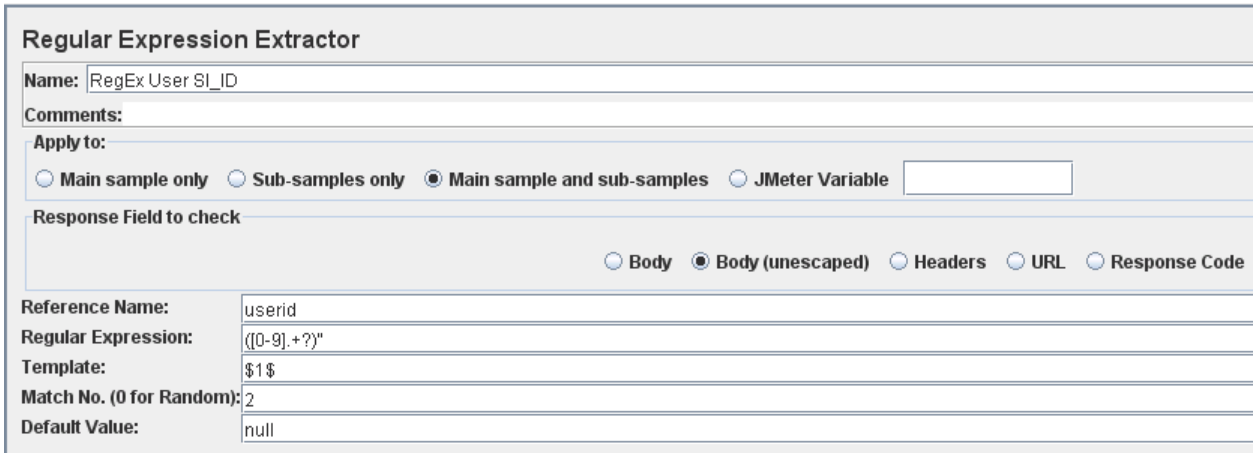
3. Next, extract the SI_ID of the current user in order to personalize the Document List.

   Select '/InfoViewApp/listing/objectList.do' in the list under 'Web Intelligence – Refresh and Drill' > '***Navigate Document List***' and Right-Click > Add >Post Processors >

Regular Expression extractor. Configure the following:

- Name: RegEx User SI_ID
- Apply to: Main sample and sub-samples
- Response Field to check: Body (unescaped)
- Reference Name: userid
- Regular Expression: ([0-9].+?)"
- Template: 1
- Match No: 2
- Default Value: null

Example:

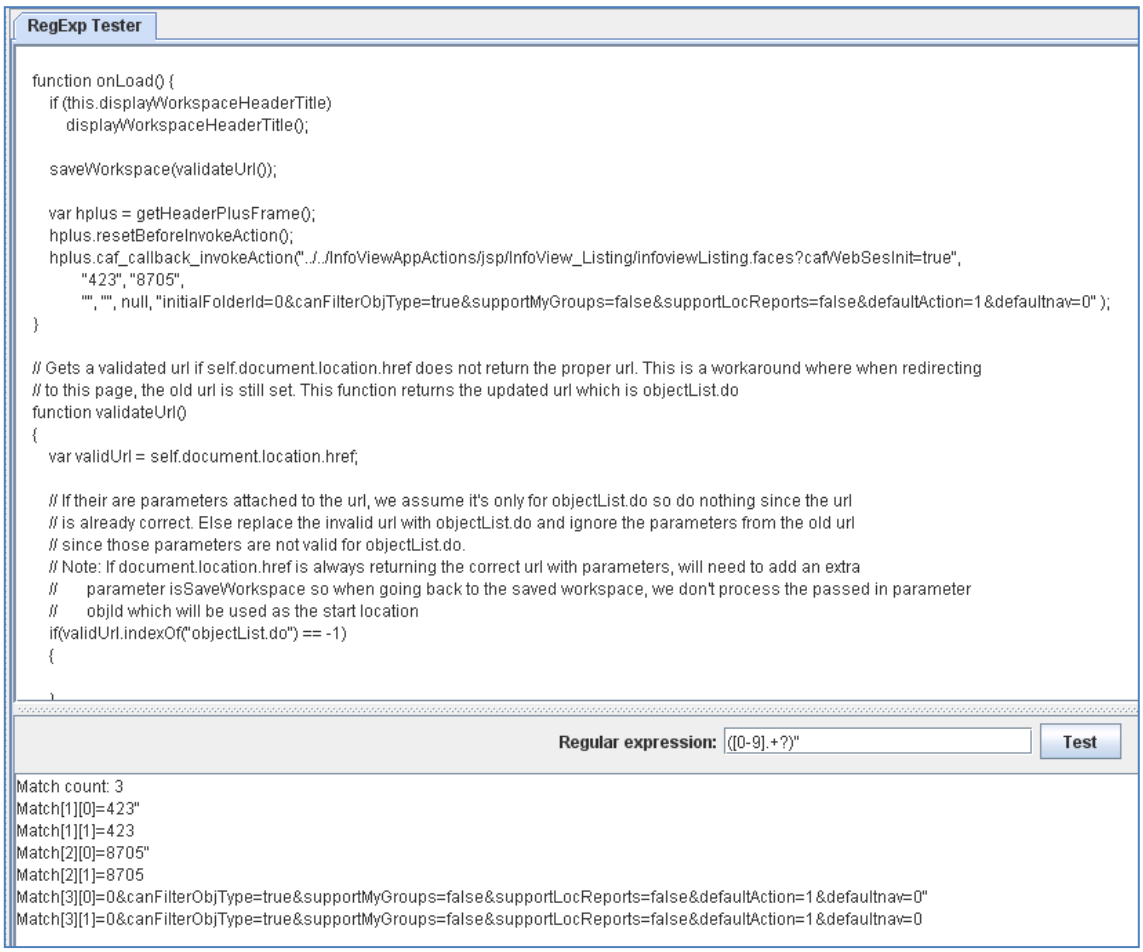


Note: The response body for this request contains multiple numeric values and we want the 2nd occurrence. Use Match No. 2 to extract the 2nd occurrence of the regular expression. You can observe this by looking at the response body of objectList.do using an HTTP debugger such as Fiddler. If no match is found, the variable will be set to a value of 'null'.

4. Pass the extracted SI_ID to the InfoView Listing face servlet.

   Select /InfoViewAppActions/jsp/InfoView_Listing/infoviewListing.faces in the list under 'Web Intelligence – Refresh and Drill' > '***Navigate Document List***' and make the following changes:

- Name: **DocList with userid**
  /InfoViewAppActions/jsp/InfoView_Listing/infoviewListing.faces
    - This, again, is a non-functional change for ease of identification
- objIds: ${userid_g1}
    - We need to identify the first variable group extracted for userid. This is done by using the syntax ${userid_g1}
    - You can test this functionality by using the RegExp function of the View Results Tree component. The View Results Tree component will be discussed at greater length at a later time, but an example might look as follows:

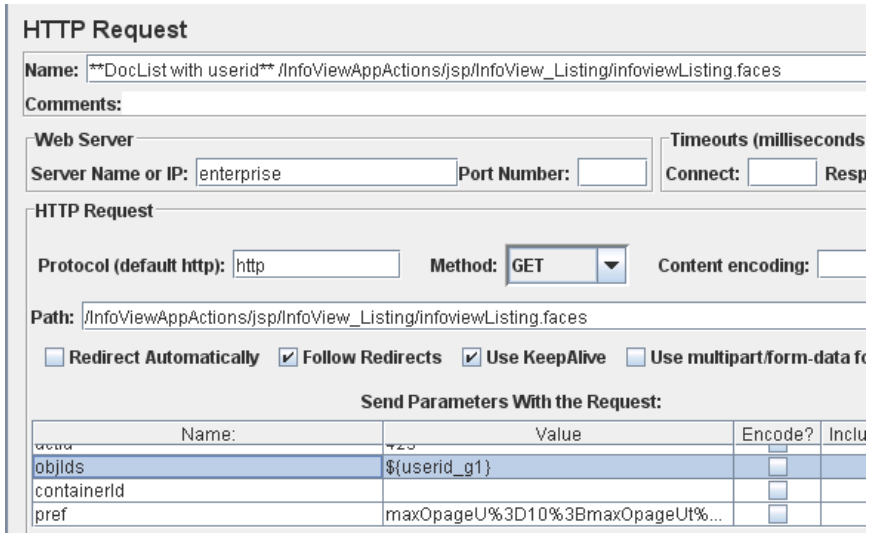


In this example, the SI_ID of the user is 8705. The regular expression stores the matches listed above and we can reference them later in our script. Since we want the 2nd match (the first match, 423, is the SI_ID of the InfoView_Listing object), we use Match No. 2 in the Regular Expression Extractor, and apply _g1 to the end of the variable reference in order to get the value without the trailing double-quote.

More information on JMeter regular expressions can be found here:

http://jakarta.apache.org/jmeter/usermanual/regular_expressions.html

An example of applying the extracted User ID to the InfoView Listing servlet:



The actions up to this point should result in a successful login, access of the InfoView Document List, and navigation to the 'Feature Samples' folder containing the report.

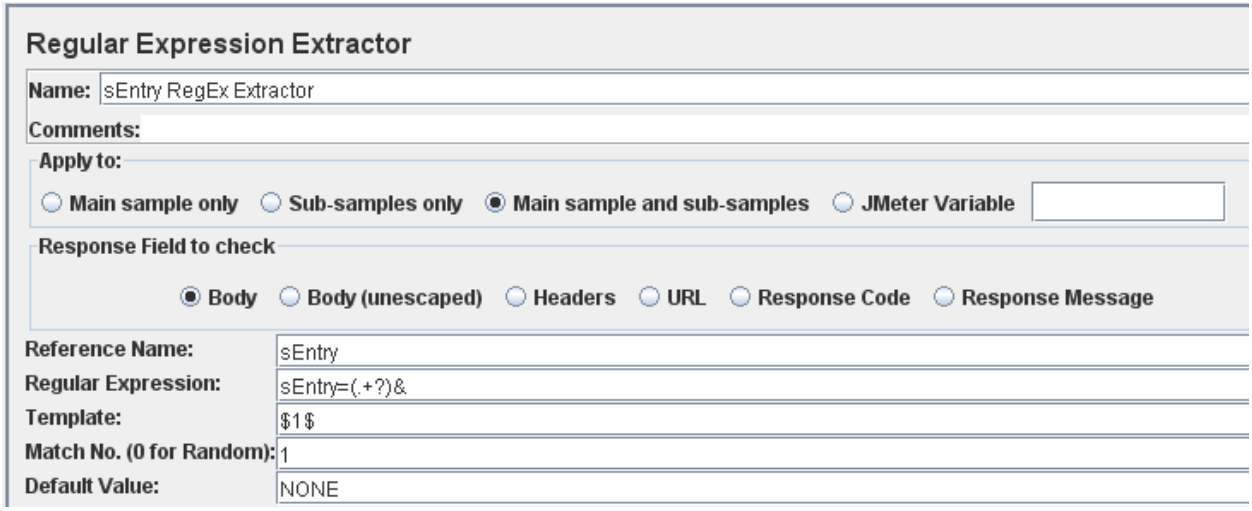
When interacting with a Web Intelligence report, it is necessary to extract a variable called 'sEntry', which represents a Web Intelligence Report Server session, and inject the value into the remaining Webi related requests.

5. Select /AnalyticalReporting/viewers/cdz_adv/viewDocument.jsp in the list under 'Web Intelligence – Refresh and Drill' > '***View and Refresh Web Intelligence Document***' and Right-Click > Add > Post Processors > Regular Expression extractor.

   Configure as follows:

   - Name: sEntry RegEx Extractor
   - Applies to: Main sample and sub-samples
   - Response Field to Check: Body
   - Reference Name: sEntry
   - Regular Expression: sEntry=(.+?)&
   - Template: 1
   - Match No: 1
   - Default Value: NONE

Example:

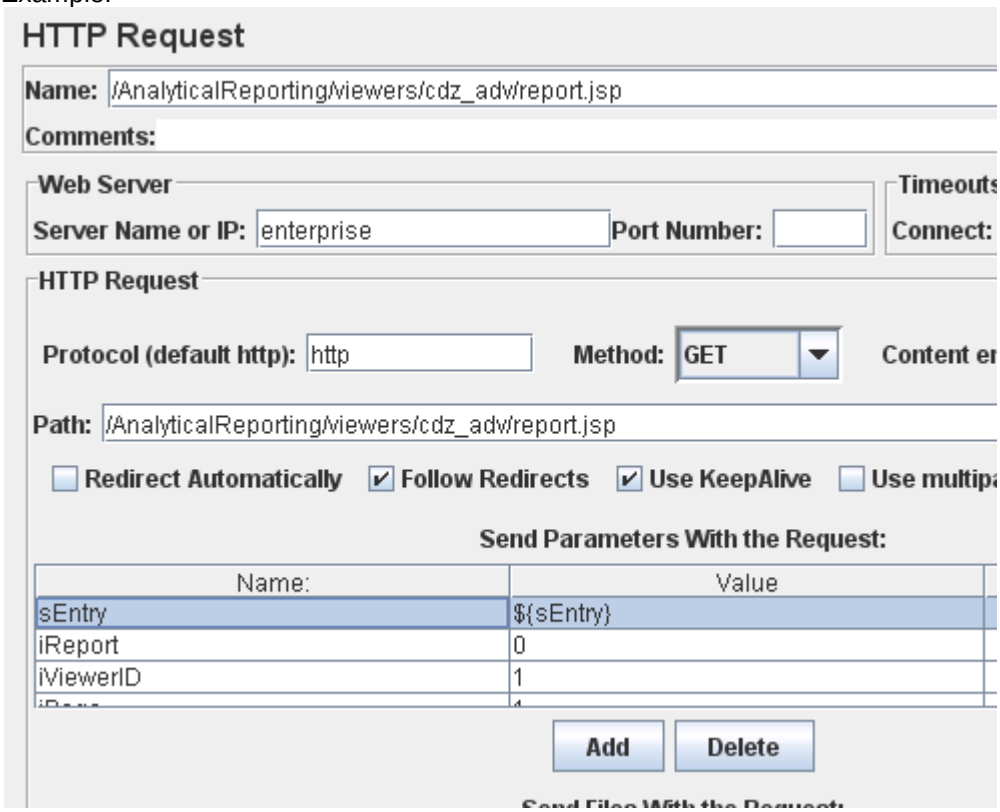


6. Find all occurrences of the following entries located under 'Web Intelligence – Refresh and Drill' > '***View Web Intelligence Document***':

```
/AnalyticalReporting/viewers/cdz_adv/report.jsp
/AnalyticalReporting/viewers/cdz_adv/viewDrillbar.jsp
/AnalyticalReporting/viewers/cdz_adv/viewReport.jsp
/AnalyticalReporting/viewers/cdz_adv/getImage.jsp
```

   Modify the sEntry parameter from the existing value to ${sEntry}

Example:



7. In order to complete the configuration of the '***View Web Intelligence Document***' transaction controller, we must also extract the names of any images that appear in the Web Intelligence report. These images comprise any graphics or logos that exist in the report, as well as some structural components of a report such as cell backgrounds defined from a file.

   Images, such as charts, that appear in a Web Intelligence Report are prefixed with TMP*, such as TMP*1*3. Images defined from a file location follow the format bores://, such as bores://00001. Finally, images that are active for drill are prefixed with the string, 'dxXMLQuickPaginated'.

   Select /AnalyticalReporting/viewers/cdz_adv/viewReport.jsp in the list under 'Web Intelligence – Refresh and Drill' > '***View and Refresh Web Intelligence Document***' and Right-Click > Add > Post Processors > Regular Expression extractor.

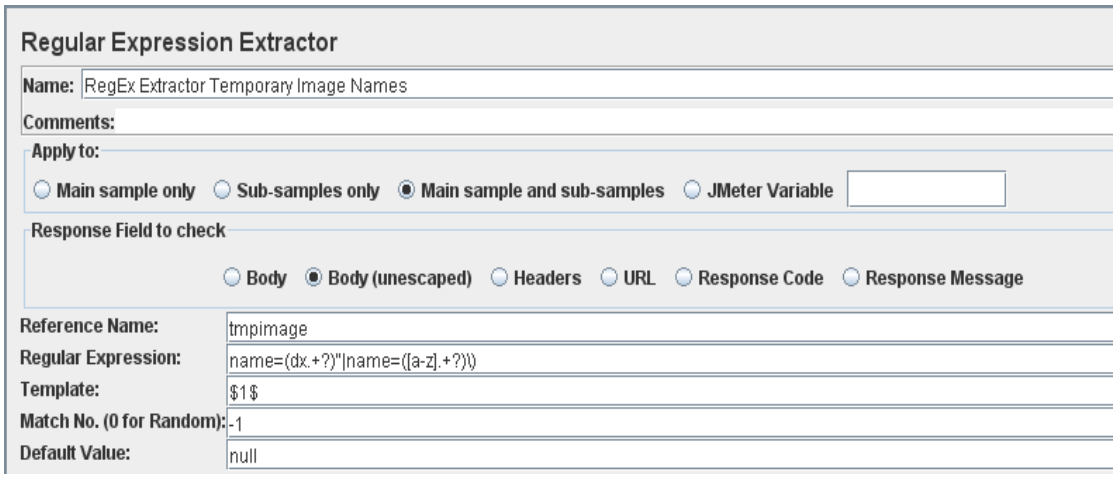
   Configure as follows:

   - `Name: RegEx Extractor Temporary Image Names`
   - `Main sample and sub-samples`
   - `Response Field to check: Body (Unescaped)`
   - `Reference Name: tmpimage`
   - `Regular Expression: name=(dx.+?)"|name=([a-z].+?)\)`

     - `This is a more complicated regular expression because the two image formats are delimited by different ending characters. The difference will be highlighted below`

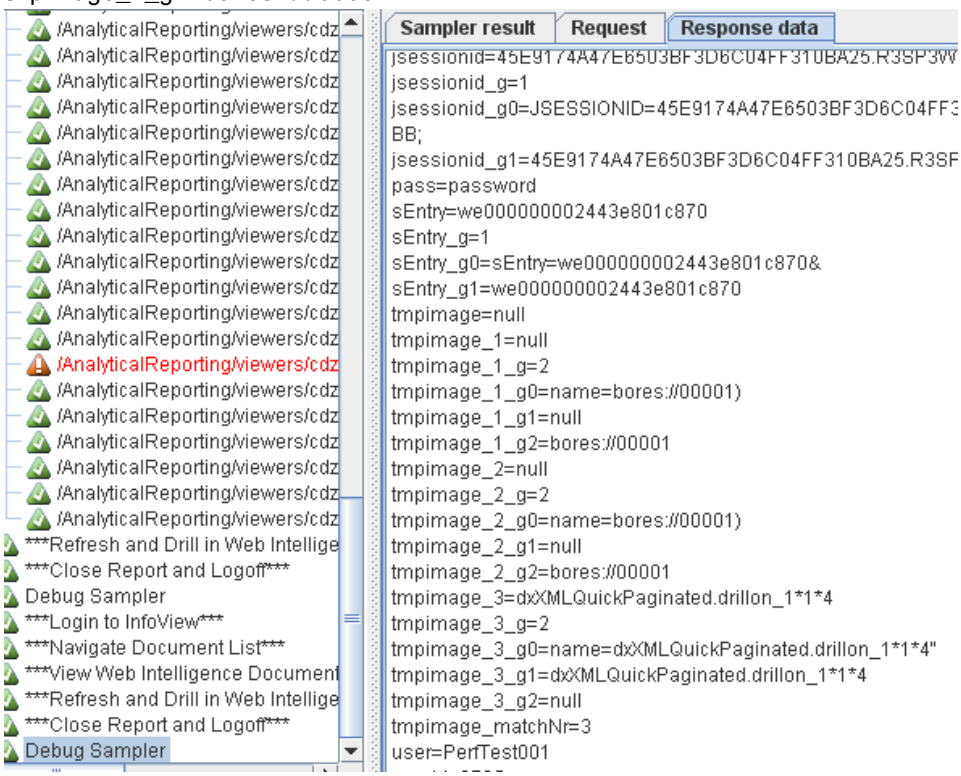
- Template: 1
- Match No: -1
    - Using -1 instructs JMeter to create variables for every match that is found. This gives us more granular control over what value to pass into each getImage.jsp call.
- Default Value: null

Example:



8. By using the 'View Results Tree' component in conjunction with the 'Debug Sampler' component, we can see the values needed for the getImage.jsp calls:

```
tmpimage_3_g1= dxXMLQuickPaginated.drillon_1*1*4
tmpimage_1_g2=bores://00001
```



The format of the temporary images differs between those used to identify drillable images (dxXMLQuickPaginated) and those beginning with "bores". Using a pipe "|" in the regular expression allows us to specify an alternate format, and returns 2 sets of matches. Any alphanumeric string prefixed by 'name=', starting with the letters 'dx' and ending with a double-quote will be captured. Alternately, any alphanumeric string prefixed with 'name=', containing lower case letters between a-z and ending in a right parentheses will be extracted.

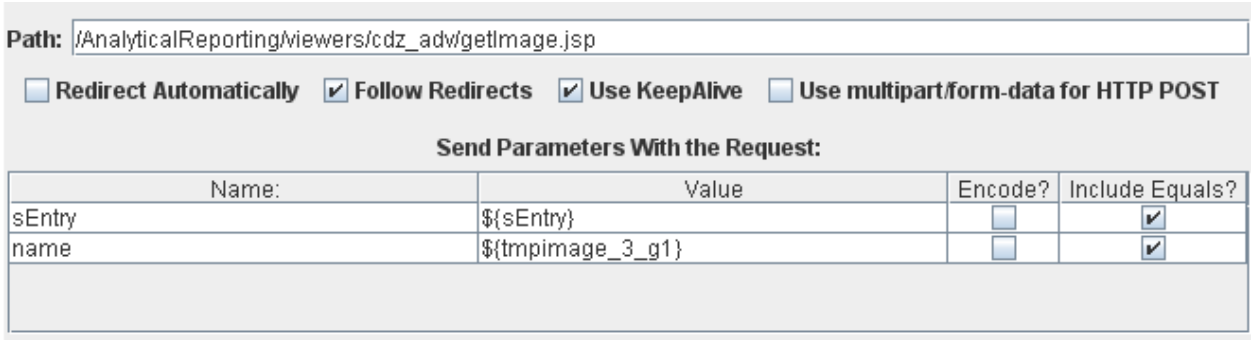
9. Select the first instance of /AnalyticalReporting/viewers/cdz_adv/getImage.jsp in the list under 'Web Intelligence – Refresh and Drill' > '***View and Refresh Web Intelligence Document***'

    - `Modify the 'name' parameter to read ${tmpimage_3_g1}`

10. Select the second instance of /AnalyticalReporting/viewers/cdz_adv/getImage.jsp in the list under 'Web Intelligence – Refresh and Drill' > '***View and Refresh Web Intelligence Document***'

    - `Modify the 'name' parameter to read ${tmpimage_1_g2}`

Example:



For every image contained in a Web Intelligence report there is a corresponding getImage.jsp request. This means that a report with sections, each section containing a graph, could have dozens of getImage.jsp requests. This can require a lot of manual effort to update each of the requests. Since images created by Web Intelligence for charts and graphs are well compressed, they have very little impact on the Webi Report Server resource consumption. It is therefore recommended that images be kept to a minimum when choosing a candidate report.

At this point, we have successfully customized the transaction controller, ***View Web Intelligence Document***, so that it creates a session, retrieves a document from the FRS, and loads it into the viewer. We're ready to perform actions on the report such as refresh, drill, page navigation, etc.

For the rest of the Test Plan, I will exclude additional example screenshots as you should now be familiar with all of the components used in this test.

11. Executing a refresh on a Web Intelligence document requires a new document state. That is, the sEntry we used above refers to the document as it exists in the Input FRS, with any saved data in the report. By refreshing, we create a new document state, where the report will contain the latest data from the database. As such, we need to extract the new sEntry, that corresponds to the document state after refresh.

    You can reduce the number of sEntry required by using reports that are marked with the property 'Refresh on Open'. This document is intended to provide insight into the functions of the BI Platform, however, so we have not used the option in this scenario.

Expand the transaction controller, ***Refresh and Drill in Web Intelligence Document***, select /AnalyticalReporting/viewers/cdz_adv/refreshDocument.jsp and Right-Click > Add > Post Processors > Regular Expression extractor.

Configure as follows:

- Name: 2sEntry RegEx Extractor
- Applies to: Main sample and sub-samples
- Response Field to Check: Body
- Reference Name: 2sEntry
- Regular Expression: sEntry=(.+?)&
- Template: 1
- Match No: 1
- Default Value: NONE

12. Select /AnalyticalReporting/viewers/cdz_adv/report.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller.

    - Apply the 2sEntry variable by changing the sEntry parameter to ${2sEntry}
    - Apply the 2sEntry variable to the following subsequent calls

      ```
      /AnalyticalReporting/viewers/cdz_adv/viewDrillbar.jsp
      /AnalyticalReporting/viewers/cdz_adv/viewReport.jsp
      /AnalyticalReporting/viewers/cdz_adv/getImage.jsp
      /AnalyticalReporting/viewers/cdz_adv/processDataFetch.jsp
      /AnalyticalReporting/viewers/cdz_adv/executeDrillAction.jsp
      ```

    - Stop at the /AnalyticalReporting/viewers/cdz_adv/executeDrillAction.jsp request. This request will generate a new sEntry request and will be handled below with an additional Regular Expression extractor.

13. Refreshing the report will also generate some new temporary images.

    Select /AnalyticalReporting/viewers/cdz_adv/viewReport.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller and Right-Click > Add > Post Processors > Regular Expression extractor.

    Configure as follows:

    - Name: 2tmpimage RegEx extractor
    - Main sample and sub-samples
    - Response Field to check: Body (Unescaped)
    - Reference Name: 2tmpimage
    - Regular Expression: name=([A-Z].+?)"|name=([a-z].+?)\)
    - Template: 1
    - Match No: -1
    - null

14. Select /AnalyticalReporting/viewers/cdz_adv/getImage.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller.
    - Apply the 2tmpimage variable to the name parameter by updating the value to ${2tmpimage_3}

- To the successive getImage.jsp request, change the name parameter to ${2tmpimage_1_g2}

15. Drilling in a document creates a new document state, and a new sEntry along with it. This allows the Webi Report Server to maintain document states throughout the course of drill up/down actions.

    If the expected duration of the refresh and drill action are lengthy, it might make sense to split the ***Refresh and Drill in Web Intelligence Document*** transaction controller into 2 independent controllers. A natural split might have /AnalyticalReporting/viewers/cdz_adv/processDataFetch.jsp as the last request in a transaction controller named ***Refresh Web Intelligence Document*** and /AnalyticalReporting/viewers/cdz_adv/executeDrillAction.jsp as the first request in a transaction controller named ***Drill in Web Intelligence Document***. In this example, the refresh and drill durations are negligible, so the two actions have been combined.

    Select /AnalyticalReporting/viewers/cdz_adv/executeDrillAction.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller and Right-Click > Add > Post Processors > Regular Expression extractor.

    Note: There will be a lengthy query string attached to the end of executeDrillAction.jsp, so the full request might look like:

    ```
    /AnalyticalReporting/viewers/cdz_adv/executeDrillAction.jsp?iReport=0&iV
    iewerID=1&sReportMode=Analysis&sPageMode=QuickDisplay&sDrillAction=yes&i
    FoldPanel=0&zoom=100&isInteractive=true&idRef=lnf.lne&sFollowBid=195719&
    sCancel=DL0
    ```

    Configure as follows:

    1. Name: 3sEntry RegEx Extractor
    2. Applies to: Main sample and sub-samples
    3. Response Field to Check: Body
    4. Reference Name: 3sEntry
    5. Regular Expression: sEntry=(.+?)&
    6. Template: 1
    7. Match No: 1
    8. Default Value: NONE

16. Select the next /AnalyticalReporting/viewers/cdz_adv/viewDrillbar.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller.

    - Apply the 3sEntry variable by changing the sEntry parameter to ${3sEntry}
    - Apply the 3sEntry variable to the following subsequent calls:

      ```
      /AnalyticalReporting/viewers/cdz_adv/viewDrillbar.jsp
      /AnalyticalReporting/viewers/cdz_adv/viewReport.jsp
      /AnalyticalReporting/viewers/cdz_adv/getImage.jsp
      ```

17. Drilling in the report will generate a final set of temporary images. In this case, the pie chart will change from displaying all states, to displaying only the state of New York (based on the drill selection recorded earlier)

    Select the next instance of /AnalyticalReporting/viewers/cdz_adv/viewReport.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller and Right-Click > Add > Post Processors > Regular Expression extractor.

    Configure as follows:

    9. Name: 3tmpimage RegEx extractor
    10. Main sample and sub-samples
    11. Response Field to check: Body (Unescaped)
    12. Reference Name: 3tmpimage
    13. Regular Expression: name=([A-Z].+?)"|name=([a-z].+?)\)
    14. Template: 1
    15. Match No: -1
    16. null

18. Select the next instance of /AnalyticalReporting/viewers/cdz_adv/getImage.jsp from the ***Refresh and Drill in Web Intelligence Document*** transaction controller.

    17. Apply the 3tmpimage variable to the name parameter by updating the value to ${3tmpimage_3}
    18. To the successive getImage.jsp request, change the name parameter to ${3tmpimage_1_g2}

19. With the drill action now complete in the Web Intelligence document, we are done modifying the ***Refresh and Drill in Web Intelligence Document*** transaction controller, and can customize the final requests needed to complete the performance test.

    Expand the ***Close Report and Logoff*** transaction controller and select the /InfoViewAppActions/jsp/InfoView_Listing/infoviewListing.faces request

    19. Modify the objIds parameter to apply the variable ${userid_g1}

20. Select the /AnalyticalReporting/viewers/cdz_adv/processClose.jsp request

    - Modify the sEntry parameter to apply the variable ${3sEntry}
    - This closes the existing Web Intelligence session that we have open

21. Because we store the JSESSIONID for each user in a cookie, there is no need to perform any additional customization to correctly logoff.

This completes the customization of dynamic data in the performance test. In order to make the test easier to view, each transaction controller can be collapsed at this time. Make sure to save the resulting Test Plan before moving on.

Trialing the Performance Test with a Debug Sampler

The performance test is now ready to execute. While it should be completely functional, there is always the possibility of an error occurring during customization. Rather, than execute a 50-100 user test that fails on login, it is advisable to first run the test for 1-2 users and attach both a Debug Sampler and View Results Tree component to the test plan.

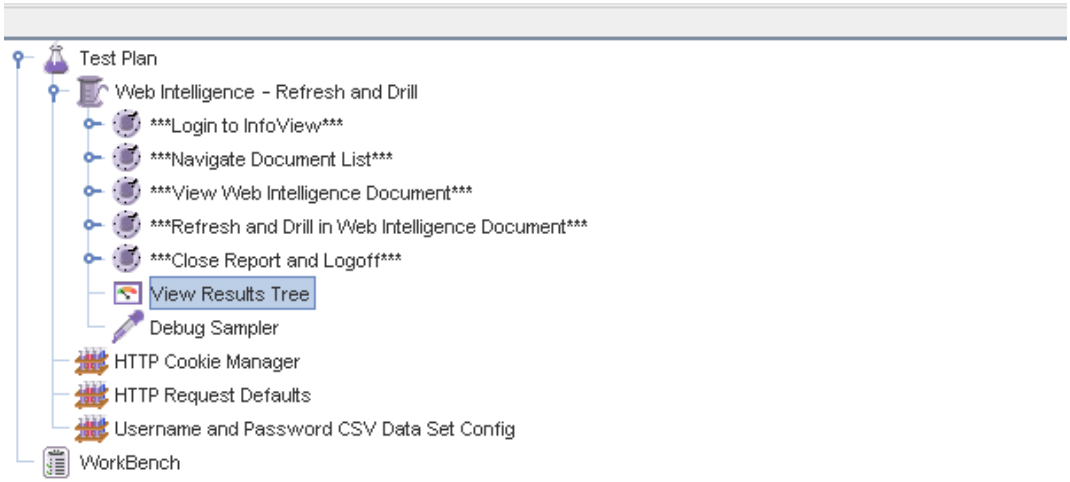
The Debug Sampler generates a sample containing the values of all JMeter variables and/or properties. We can use the Debug Sampler to ensure the Regular Expression Extractors are matching appropriate values for use with dynamic data, and the View Results Tree component to examine how the server responds to each request in the Test Plan. These components are best used in the debugging of a test, as they can be quite resource intensive when used with numerous users.

1. To attach the Debug Sampler to your test plan, use the following step:

   - Select the Thread Group, Web Intelligence – Refresh and Drill, and Right-Click > Add > Sampler > Debug Sampler

2. To attach the View Results Tree component, use the following step:
   - Select the Thread Group, Web Intelligence – Refresh and Drill, and Right-Click > Add > Listener > View Results Tree

Example:



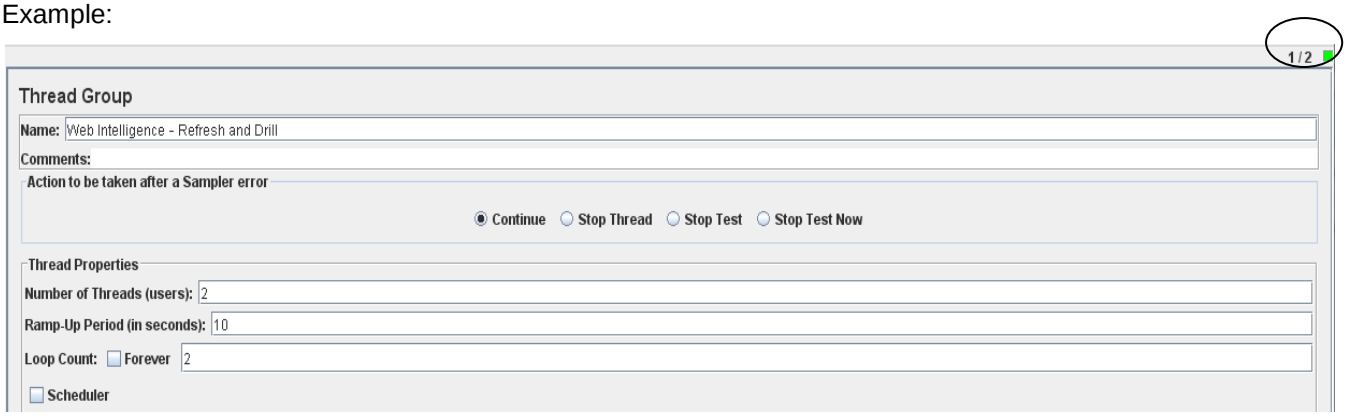
3. To configure the appropriate number of threads for the first execution, select 'Web Intelligence – Refresh and Drill' and select the following options:

   20. Action to be taken after a Sampler error: Continue
   21. Number of Threads (users): 2
   22. Ramp-Up Period (in seconds): 10

   23. This is the amount of time it will take for all of our threads to be active. That is, in 10 seconds there will be 2 active users. This setting is especially important in large tests to avoid overwhelming the client or server with too many simultaneous requests

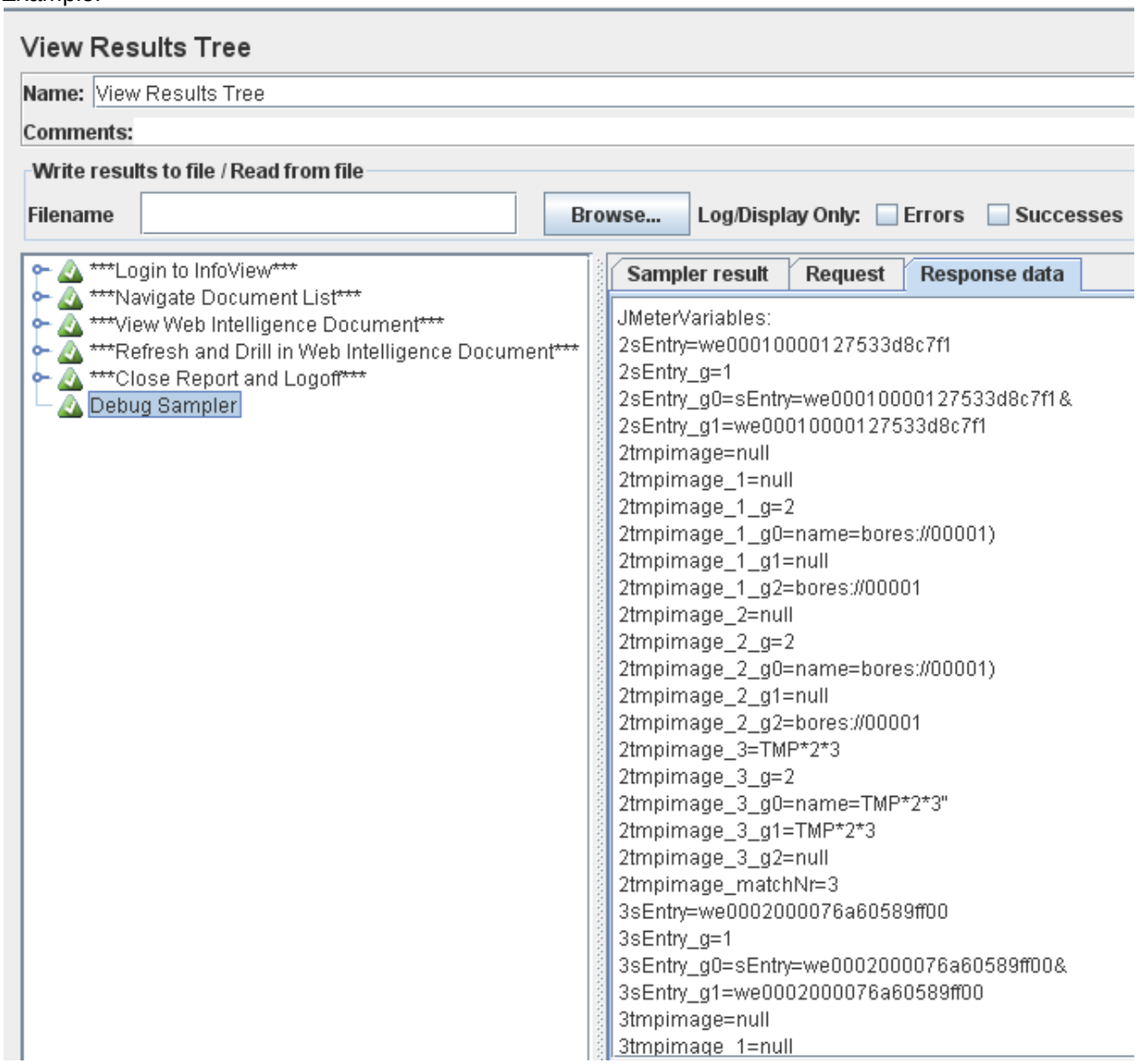
   24. `Loop Count: 2`

Example:



4. Execute the test by selecting ‘Run’ > ‘Start’ from the top menu
5. The box in the top right corner should begin to increment up from 0/2 to 2/2 and turn green
6. Select the ‘View Results Tree’ component added previously

Example:



7. If everything has been configured properly the tree view should show a series of green checkmarks. An error would show up as a red exclamation point.
8. Error handling is a complex process that warrants a separate guide. If this test returns an error, it is most likely a typo somewhere, so expand the relevant branch in the tree and identify the problematic request. The response data should provide enough detail for you to refer back to the guide and address any inconsistencies.

Results Analysis

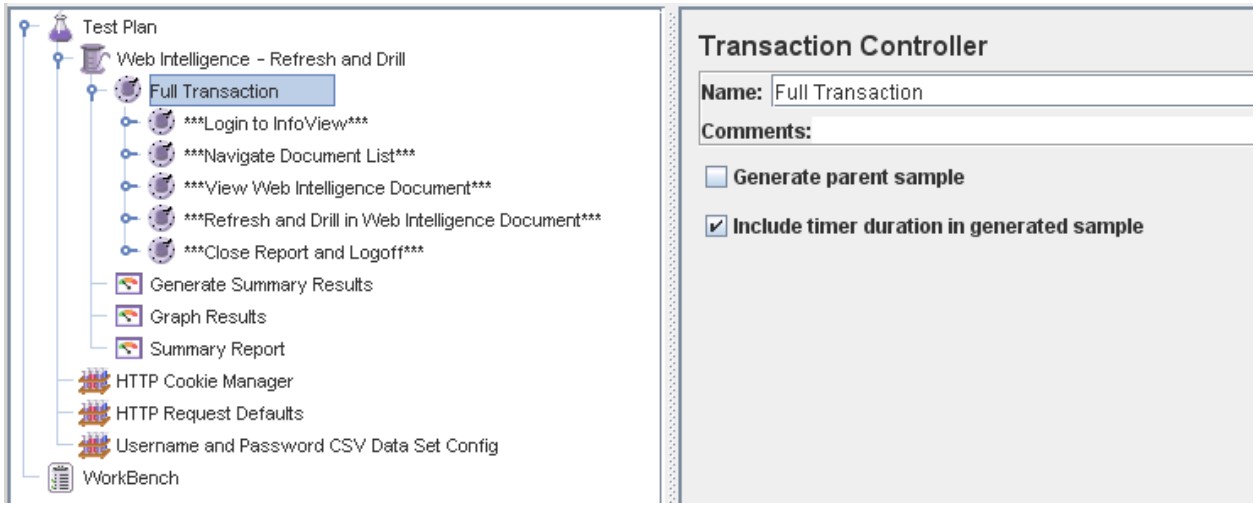
After successfully completing a trial run of the performance test, we're ready to run a complete iteration and collect the corresponding result data.

In order to simplify our results analysis, we can add an additional transaction controller called 'Full Transaction' and place the other controllers beneath it. This allows us to generate sample timers for each component of the test as well as the entire test. To create the final transaction controller, follow these steps:

1. Right-click the Thread Group 'Web Intelligence – Refresh and Drill' > Add > Logic Controller > Transaction Controller
2. Starting with '***Login to InfoView***' drag and drop each of the 5 named transaction controllers onto the new object and select 'Add as Child'
3. Select the newly created transaction controller, and configure the following properties:

    - Name: Full Transaction
    - Disable the option 'Generate Parent Sample'
    - Enable the option 'Include timer duration in generated sample'

The final result should look something like this:



By reviewing this data, along with performance data from the physical machines running SAP BusinessObjects Enterprise XI 3.1, we can determine whether or not the objective has been met.

The objective:

Determine how many CPU are required for Web Intelligence to process 50 concurrent report refresh and drill requests (of the Interactive Analysis report) with an average response time not exceeding 90 seconds and maximum response time not exceeding 120 seconds.

To configure and execute the complete performance test, configure the following properties:

1. Remove the 'View Results Tree' component by highlighting it and pressing 'Delete'.
2. Remove the 'Debug Sampler' component by highlighting it and pressing 'Delete'.
3. Select the Thread Group 'Web Intelligence – Refresh and Drill' and modify the settings accordingly:

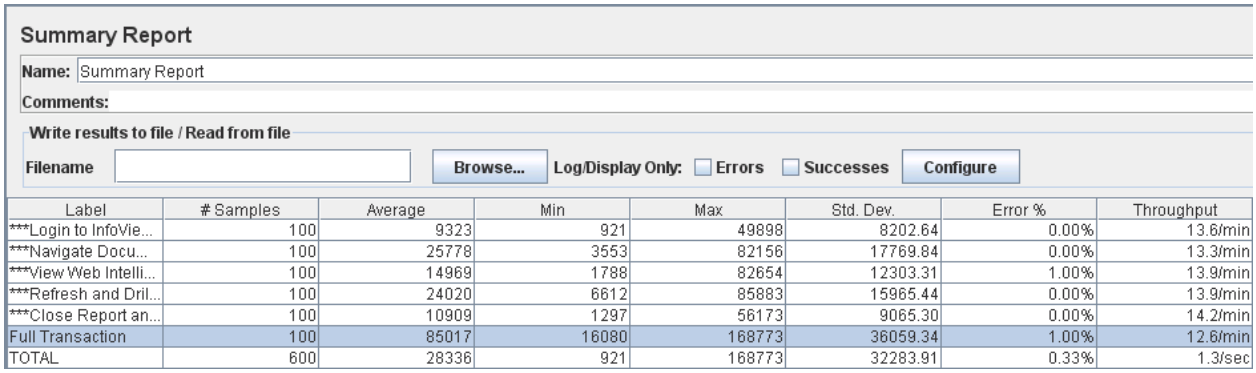
   - `Number of Threads (users): 50`
   - `Ramp-Up Period (in seconds): 300`
   - `Loop Count: 2`

   These settings execute the test for 50 users, who will be fully active in the system after 5 minutes. We repeat the test once to ensure the necessary concurrency is present on the system. This results in one new user workflow being started every 6 seconds (300 seconds / 50 users = 6 seconds per thread).

4. Right-click the Thread Group 'Web Intelligence – Refresh and Drill' > Add > Listener > Graph Results
5. Right-click the Thread Group 'Web Intelligence – Refresh and Drill' > Add > Listener > Summary Report

   The 'Graph Results' component allows us to chart response time and throughput over the course of the test. The 'Summary Report' component creates a summary of the test in table format.

6. Run the test by selecting 'Run' > 'Start'
7. Once the test is complete, select the 'Summary Report' component from the Test Plan:

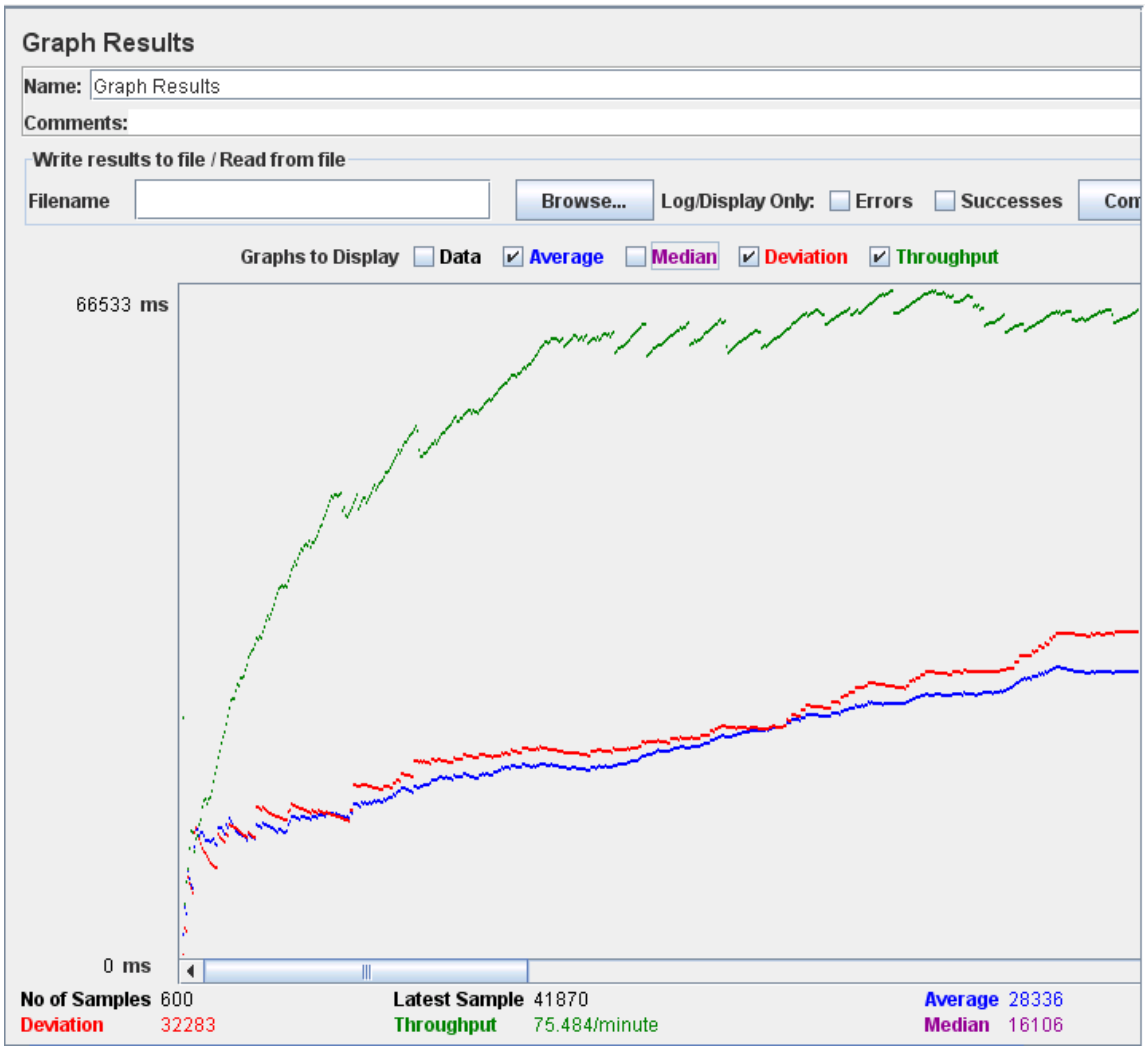


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput
***Login to InfoVie...	100	9323	921	49898	8202.64	0.00%	13.6/min
***Navigate Docu...	100	25778	3553	82156	17769.84	0.00%	13.3/min
***View Web Intelli...	100	14969	1788	82654	12303.31	1.00%	13.9/min
***Refresh and Dril...	100	24020	6612	85883	15965.44	0.00%	13.9/min
***Close Report an...	100	10909	1297	56173	9065.30	0.00%	14.2/min
Full Transaction	100	85017	16080	168773	36059.34	1.00%	12.6/min
TOTAL	600	28336	921	168773	32283.91	0.33%	1.3/sec

The image above indicates each transaction was executed 100 times (50 users and 2 iterations), that the average response time was ~85 seconds (85017 ms), with a maximum duration of ~168 seconds (168773)

We can also tell that the two longest running functions, on average, were the 'Navigate Document List' at ~25.7 seconds, and the 'Refresh and Drill in Web Intelligence Document' at ~24 seconds.

8. Next, select the 'Graph Results' component from the Test Plan:



From the graph above, we see that average response time (indicated by the blue line) increases over time and that throughput levels off as load increases. You can also see that the value for Average (28336) corresponds to the TOTAL average in the 'Summary Report' as opposed to the 'Full Transaction' timer, so the actual numbers are skewed. To chart only the 'Full Transaction' timer, you would need to disable the 'Generate Parent Sample' option on the nested controllers such as 'Login to InfoView'.

By reviewing the 'Summary Report' we can determine that the average response time criteria we set of 90 seconds has been met (Average Response time was ~85 seconds) but that we exceeded the maximum response time of 120 seconds. There is a gradual increase to response time as the test progressed, as evidenced by the graph above. We would need to review performance data corresponding to the Application Server, Intelligence Tier, and Processing Tier, as well as relevant Database statistics and try to identify any potential bottlenecks, such as CPU consumption, and modify the parameters of the test before running it again. These concepts are out of scope for this particular document but will be covered in a supplemental guide.

The intent of this document is to demonstrate the key features of Apache JMeter and how they pertain to creating performance tests for SAP BusinessObjects Enterprise XI 3.1 and Web Intelligence. There are many more features available in the tool for extending and fine tuning performance tests, and the tool can be adapted to a wide variety of uses.

After completing the exercise in this document, you should have a fully functional performance test leveraging one of the report samples that ships with SAP BusinessObjects Enterprise XI 3.1. You can use this methodology to create your own performance tests based on a variety of reports and business processes.

Related Content

http://www.sdn.sap.com/irj/sdn/index

For more information, visit the Business Objects homepage.

Copyright