

**Oracle® Insurance Policy
Administration**

Load Testing Methods

Version 9.7.0.0

Part number: E39062_01

May, 2013

Copyright © 2009, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are “commercial computer software” or “commercial technical data” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Where an Oracle offering includes third party content or software, we may be required to include related notices. For information on third party notices and the software and related documentation in connection with which they need to be included, please contact the attorney from the Development and Strategic Initiatives Legal Group that supports the development team for the Oracle offering. Contact information can be found on the Attorney Contact Chart.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

CONTENTS

Preface

- v Audience
- v Documentation Accessibility
- v Conventions

Chapter 1: Load Testing Oracle Insurance Policy Administration

2 Introduction to Performance Testing

- 2 Need for Performance Testing
- 2 Types of Performance Testing
- 3 Performance Testing Objectives

5 OIPA Product Overview

- 5 OIPA Architecture
 - 6 The Business Rules
 - 6 The Transactions
 - 6 The User Data

9 Load Testing OIPA

- 9 Load Testing OIPA Using Apache JMeter
 - 9 Introduction to Apache JMeter
 - 9 Benefits of Using Apache JMeter
 - 10 Load Testing OIPA Using Apache JMeter
- 21 Load Testing OIPA Using NeoLoad
 - 21 Introduction to NeoLoad
 - 21 Benefits of Using NeoLoad
 - 22 Load Testing OIPA Using NeoLoad

Preface

This chapter includes the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

AUDIENCE

This document is intended for professionals involved in testing the performance of OIPA.

DOCUMENTATION ACCESSIBILITY

This section includes the following topics:

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

CONVENTIONS

The following text conventions are used in this document:

Convention	Description
bold	Boldface type indicates graphical user interface elements associated with an action.

Convention	Description
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Chapter 1

Load Testing Oracle Insurance Policy Administration

This module describes performance testing and the procedure to run load tests on OIPA by using Apache JMeter and NeoLoad. This module also describes the OIPA architecture because it is important to understand the various parts of the OIPA application on which testing is performed.

This module includes the following topics:

- *Introduction to Performance Testing*
- *OIPA Product Overview*
- *Load Testing OIPA*

INTRODUCTION TO PERFORMANCE TESTING

Performance testing is the process of determining the speed at which an application processes input to produce output. Performance testing can involve quantitative tests performed in a lab, such as measuring the response time or the number of instructions per second at which an application functions. Qualitative attributes such as reliability, scalability, and interoperability can also be part of performance test criteria. The results of performance testing can be used to tune and scale an application.

This section includes the following topics:

- *Need for Performance Testing*
- *Types of Performance Testing*
- *Performance Testing Objectives*

NEED FOR PERFORMANCE TESTING

Performance testing can serve different purposes. Some of the purposes for doing performance testing are:

- Demonstrate that the system meets performance criteria
- Removes bottlenecks and helps setting up a baseline for future regression testing
- Optimize the most important application performance trait, and user experience
- Check whether all the test cycles conform to the test plan
- Check whether performance acceptance criteria are met by the system
- Collect the performance metrics
- Determine the performance characteristics for various configuration options
- Ensure the stability of the system being tested

TYPES OF PERFORMANCE TESTING

Several types of performance testing must be used to fully gauge the behavior and response time of a system during regular usage. In order to test a system prior to deployment, system usage is simulated using a variety of specialized software packages. Most commonly, the following types of performance tests are used:

- *Load*
- *Endurance*
- *Spike*
- *Scalability*

Load

Load testing is used to measure the behavior and response times of the system during a specified amount of utilization. In order to load test a system, system utilization is simulated using specialized load testing software. In addition to testing the system for regular utilization loads, systems are also stress tested. Stress testing is the use of load testing to simulate an abnormally high, or peak, system utilization.

Endurance

Endurance testing, also known as soak testing, is used to determine if the system can sustain a continuous load over a specified period of time. Endurance testing is an important testing technique that can be used to identify issues such as memory leaks and other possible issues that could lead to performance duration in a production environment.

Spike

Spike testing is used to gauge the system performance when the load placed on the system is highly variable. The behavior and response times of the system are measured.

Scalability

Scalability testing is used to identify possible issues that may result in scaling the system. The ability of the system to adjust, based upon load demands and available resources, are tested to ensure resources are efficiently utilized.

Note This module describes only how to perform load testing on OIPA.

PERFORMANCE TESTING OBJECTIVES

This section describes the objectives of Performance testing. The following are the objectives of Performance testing:

- *Infrastructure Tuning*
- *Database Tuning*
- *Pinpointing Problem Areas*
- *Growth Needs*

Infrastructure Tuning

Infrastructure tuning is undertaken to measure the following performance:

- Network performance
- How is the data stored
- Co-location of servers

Database Tuning

Database tuning is performed in indexes, tablespaces, administrative settings, and SQL statement origination areas of the system.

Indexes

The following are some points to be considered while tuning a database:

- Ensure that the tables are not over indexed or under indexed
- Indexing must be case-insensitive
- Functional indexing is preferred to column indexing

SQL Statement Configuration

SQL statements can originate from two places in the OIPA - configuration or Java code.

SQL Statements in business rules can be complex and business logic heavy, or simple lookup queries. SQL statements originating from configuration not done by performance (mostly). These statements can be tuned via index optimization.

SQL Statements originating from the Java code are generated by the persistence layer (TopLink). These statements mostly perform basic Create, Read, Update, and Delete (CRUD) operations and some complex queries. The SQL statements originating from the Java code are optimized by Toplink and can only be tuned with indexing.

Pinpointing Problem Areas

The following problem areas are tested:

- Configuration
- Activity verification
- Code

Growth Needs

The results of performance tests can be used to plan the systematic growth of an application and the resources used by the application. The application can be scaled according to the requirements of the organization. However, keep in mind that performance testing must be performed each time the application is modified. The results obtained from running tests on a modified application might be significantly different from earlier results.

OIPA PRODUCT OVERVIEW

The Oracle Insurance Policy Administration (OIPA) system is a highly agile application that can be deployed across a numerous technology stacks. This flexibility allows you to align with your enterprise's technology requirements or best practices while implementing a state-of-the-art, next generation rules based administration system. This is a system that is unhampered by the limitations of legacy technologies and offers extensibility that maximizes system life span. It is an open architecture developed completely in Java which easily integrates into your organization's current structure.

OIPA ARCHITECTURE

OIPA is a multilayer enterprise application with the presentation, application processing and data management logically separated by different processes. This structure provides great flexibility as layers can be added or updated without affecting other layers, but also adds complexity when initially leaning how the system behaves.

Figure 1 illustrates the OIPA application, its database, the supporting applications separately, and the inside of the architectural layers.

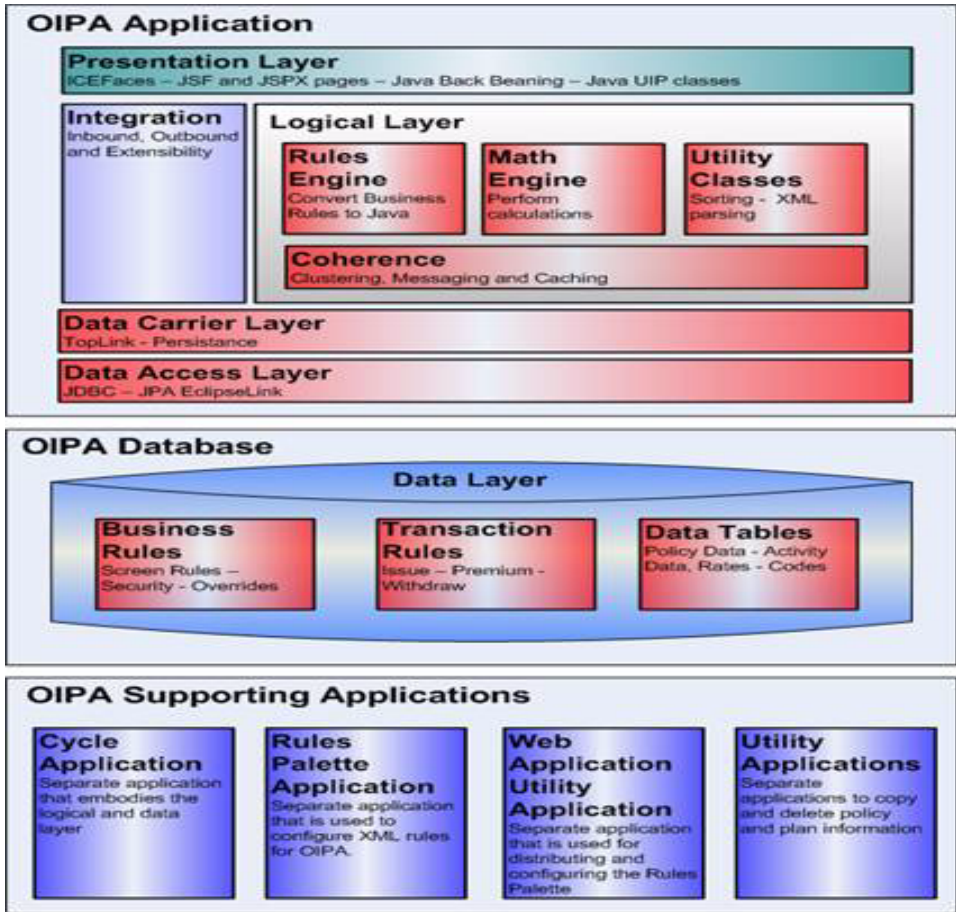


Figure 1: The OIPA Architecture

The OIPA presentation layer, which controls the display of information to the user, is built on an ICEfaces an open source AJAX Framework. ICEfaces enables rich interactive enterprise development in Java, which is completely transparent to the user.

The logical layer, which includes the actual business logic through the Rules Engine and Math Engine, is separate from presentation layer. The Rules Engine processes the configurable XML business rules stored in the database which information is then parsed into Java for compilation. and control system behavior. OIPA makes use of Oracle Coherence for clustering, messaging and caching.

The data access layers are used to retrievedata from the data carrier layer that houses the OIPA database. Oracle TopLink is used for persistence, as well as providing the framework for relationally mapping Java objects into XML. The JDBC API is used for connectivity between the database and the application.

The OIPA database is composed of the following three main areas:

The Business Rules

The Business Rules are configurable XML rules that control the behavior of the application and get processed by the Rules Engine when invoked by the user in the presentation layer. These rules control things such as security, what is on the OIPA screens, the design of insurance products and other pertinent information.

The Transactions

The Transactions are configurable XML rules that are used to execute business processes and logic. They may be run by users through the presentation layer or by the separate cycle application that is used for nightly batch processing.

The User Data

The actual user data includes all data entered into the system, such as policy and client data.

The following four applications are used in conjunction with OIPA:

The Rules Palette

The Rules Palette is a graphical editor that is used to configure business rules and transactions.

The Palette Web Application Utility

The Palette Web Application Utility is used to both distribute and configure the Rules Palette.

Cycle

Cycle is a distributed application that is used to process pending transactions in batch form.

A Utility Application

A utility application is an unsupported application. However, this application allows for manipulation of the OIPA database for tasks such as creating and deleting policy, and plan data, as well as bulk policy creation for use with performance testing.

Conceptually the OIPA architecture includes the following layers:

Presentation

The Presentation comprises the user interface, which is a part of the ICEfaces framework. Within its framework, ICEfaces caches the user data in its own container. This caching framework is designed to reduce the amount of database interaction.

Integration

In the Integration layer, `AsFile` is used for `Inbound` and `MathVariable` process for outbound and extensibility.

Persistence

This layer is used to store data while application is not running. In OIPA, `Toplink` is used for this layer.

Database

One of the major differentiators of the OIPA system is the rules-based architecture. The business and transaction rules are easily configurable XML files that are stored in the database and can be modified without touching the underlying code. This drastically reduces the time it takes to buildout the administrative support system for a product. Policy data is also stored in descriptive tables to help identify data needed for downstream or reporting.

Figure 2 shows the conceptual architecture of the OIPA.

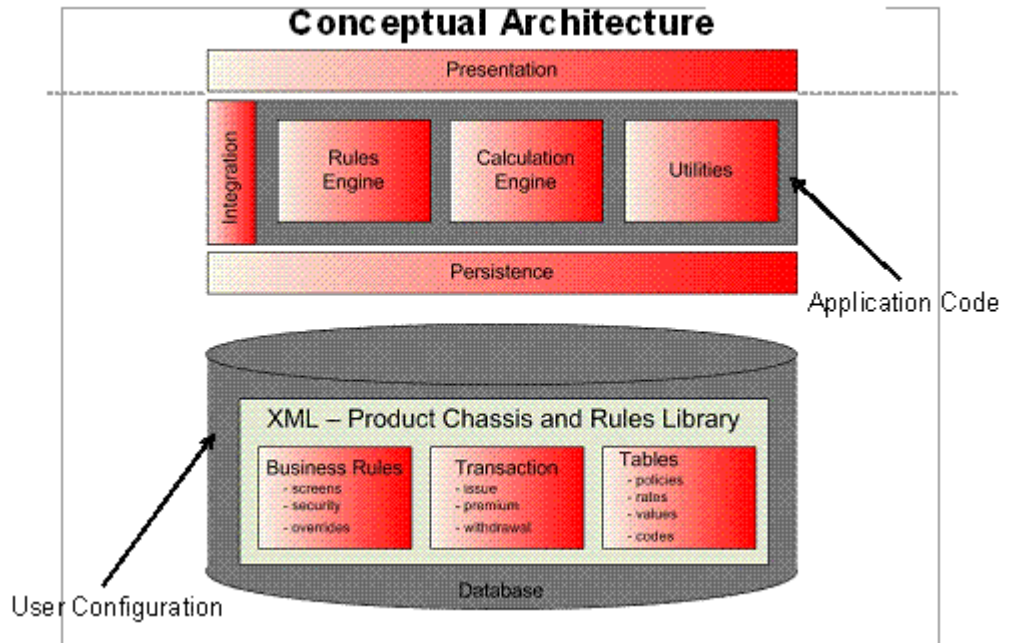


Figure 2: The Conceptual Architecture of OIPA

Note For information about the ICEfaces architecture, refer to:
http://www.icefaces.org/docs/v1_8_2/htmlguide/devguide/sys_architecture.html#1000126

LOAD TESTING OIPA

This section describes load testing Oracle Insurance Policy Application using Apache JMeter and NeoLoad. This section includes the following topics:

- *Load Testing OIPA Using Apache JMeter*
- *Load Testing OIPA Using NeoLoad*

LOAD TESTING OIPA USING APACHE JMETER

This section describes performance testing OIPA using Apache JMeter. This section includes the following topics:

- *Introduction to Apache JMeter*
- *Benefits of Using Apache JMeter*
- *Load Testing OIPA Using Apache JMeter*

Introduction to Apache JMeter

Apache JMeter is an open-source, Java desktop application. It can be used to perform load tests, assess functional behavior, and measure performance.

Benefits of Using Apache JMeter

The following are some of the benefits of using JMeter for performance testing:

- Can load and performance test many different types of servers
- Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups
- Careful GUI design allows faster operation and more precise timings
- Caching and offline analysis/replaying of test results
- Highly extensible:
 - Pluggable samplers allow unlimited testing capabilities
 - Several load statistics may be chosen with pluggable timers
 - Data analysis and visualization plug-ins allow great extensibility as well as personalization
 - Functions can be used to provide dynamic input to a test or provide data manipulation
 - Scriptable Samplers (Bean Shell is fully supported and there is a sampler which supports BSF-compatible languages)
- Best open-source tool that has the flexibility to create repeatable tests for the ICEfaces AJAX framework

Load Testing OIPA Using Apache JMeter

This section describes the steps involved in performance testing OIPA using JMeter. This section includes the following topics:

- *Adding Configuration Elements*
- *Recording*
- *Converting Fields to Parameters*
- *Extracting ICEfaces Session Variables*
- *Adding Listeners*
- *Adding Timers*
- *Altering Heap Size*
- *Defining Scope*

Adding Configuration Elements

The first step is to add Configuration Managers to the scenario.

To add elements:

1. Start Apache JMeter, and then right-click **TEST PLAN**.
A menu is displayed.
2. On the menu, click **Add, Config Element**, and then **HTTP Request Defaults**.
The HTTP Request Defaults section is displayed.
3. Enter the following details in the HTTP Request Defaults section:
 - In the Server Name field, enter the server name.
 - In the Port Number field, enter the port number.
 - In the Protocol field, enter the protocol.
 - In the Path field, enter the path of the Login screen.
4. Right-click **TEST PLAN**.
A menu is displayed.
5. On the Menu, click **Add, Config Element**, and then **HTTP Cookie Manager**.
The HTTP Cookie Manager section is displayed.
6. Ensure that the Clear cookies each iteration? check box is selected, as shown in Figure 3.

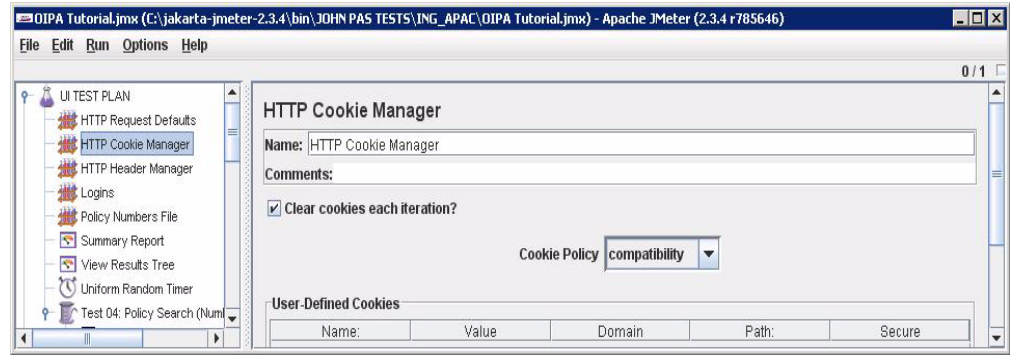


Figure 3: The HTTP Cookie Manager Section

7. Right-click **TEST PLAN**.
A menu is displayed.
8. On the menu, click **Add, Config Element**, and then **HTTP Header Manager**.
The HTTP Header Manager section is displayed.
9. In the HTTP Header Manager section, retain the default values, as shown in Figure 4.

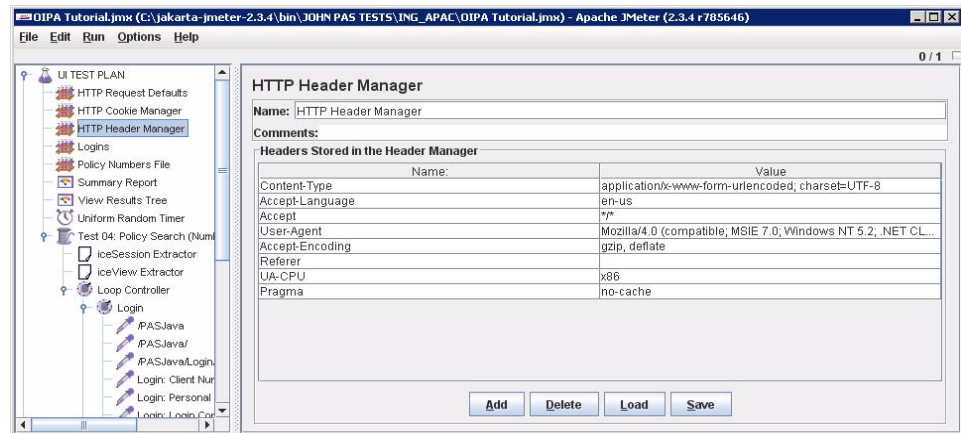


Figure 4: The HTTP Header Manager Section

10. Right-click **TEST PLAN**.
A menu is displayed.
11. On the menu, click **Add**, and then **Thread Group**.
The Thread Group section is displayed.
12. Enter the details of the thread properties in the Thread Group section.

Recording

The next step in the process is recording the activity.

To begin scripting the test case:

1. Right-click the **Workbench** element.

A menu is displayed.

2. On the menu, click **Add, Non-Test Elements**, and then **HTTP Proxy Server**.
Ensure that the port listed in the proxy server corresponds to that of your Web browser. The HTTP Proxy Server section is displayed.
3. Exclude unwanted response elements, as shown in Figure 5.

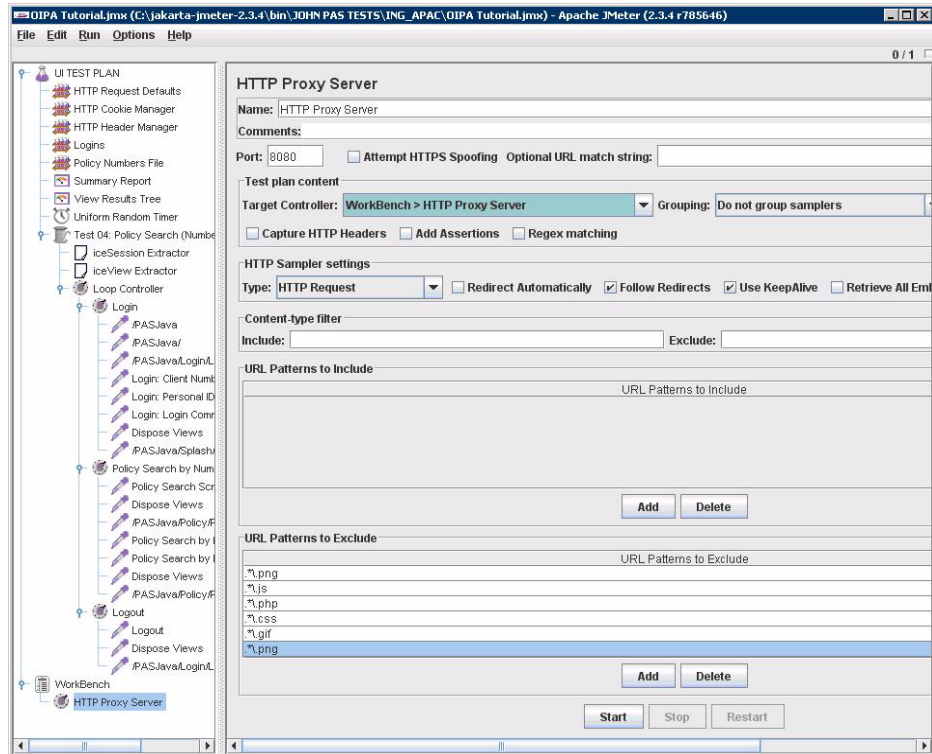


Figure 5: The HTTP Proxy Server Section

4. Click **Start**.
5. Open Internet Explorer version 7.
6. From the Tools menu, click **Internet Options**.
The Internet Options dialog is displayed.
7. Click the **Connections** tab, and then click **LAN settings**.
The Local Area Network (LAN) Settings dialog is displayed, as shown in figure 6.

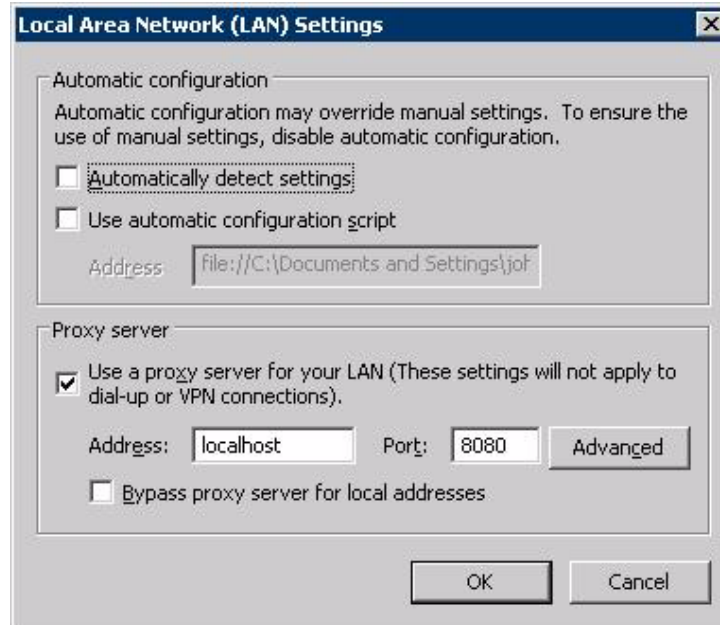


Figure 6: The Local Area Network (LAN) Settings Dialog

8. Open the desired URL, and begin scripting.
9. Log into Apache JMeter, and then open the Policy Search screen.
10. Search for a policy, open the result, and then log out of Apache JMeter.
11. Create a Thread Group, and copy all of the resulting pages to it, as shown in Figure 7.

In the thread group, you can edit the number of users, and handle execution loop.

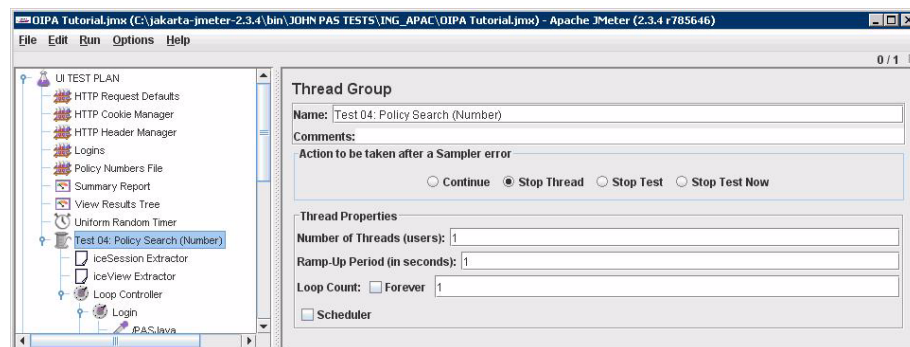


Figure 7: Creating a Thread Group

12. Add controllers, group, and rename pages to make the reports more clear.

Figure 8 shows the result of the test case, after making the naming changes and grouping the pages for clarity.

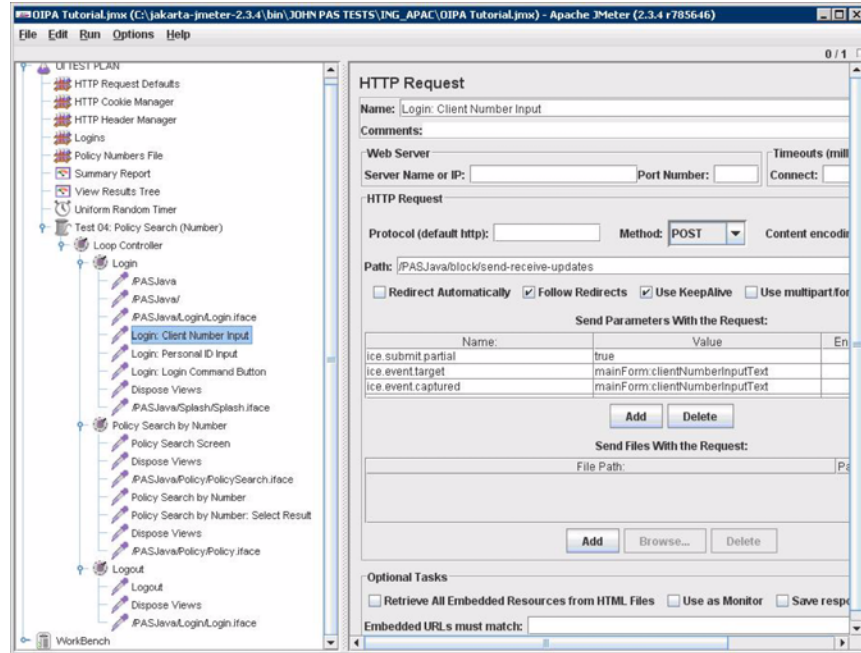


Figure 8: Result of the Test Case

Converting Fields to Parameters

In some cases, test scenarios require certain fields to be parameterized with input files, to allow for multiple users to log in, or multiple policy/client criteria for searching.

To convert field to parameters:

1. For an example, create a simple text file Logins.txt containing login information corresponding to valid username and password combinations in the system.
2. Click **Add, Config Element**, and then **CSV Data Set Config** to add an input field.

The CSV Data Set Config section is displayed.

3. In the **Filename** field, enter the name of text file you created. In this example, enter **Logins.txt**.
4. In the **Variables** field, enter appropriate variable names. In this example, enter **clientnumber, password**.

These variables will correspond to the entries contained in the Logins.txt file.

Figure 9 shows the CSV Data Set Config section with all the fields populated.

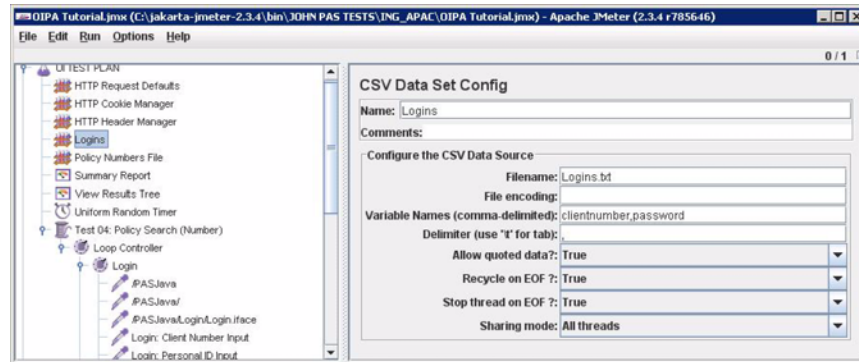


Figure 9: The CSV Data Set Config Section

5. Open the page where the variables are used. In this example, open the Login command button.
6. Replace the recorded username and password with `${clientnumber}` and `${password}`. Now, when testing, the values in the input file will be used instead of what was initially recorded, as shown in Figure 10.

Ensure that you check if the values are used in any other pages and replace where necessary.

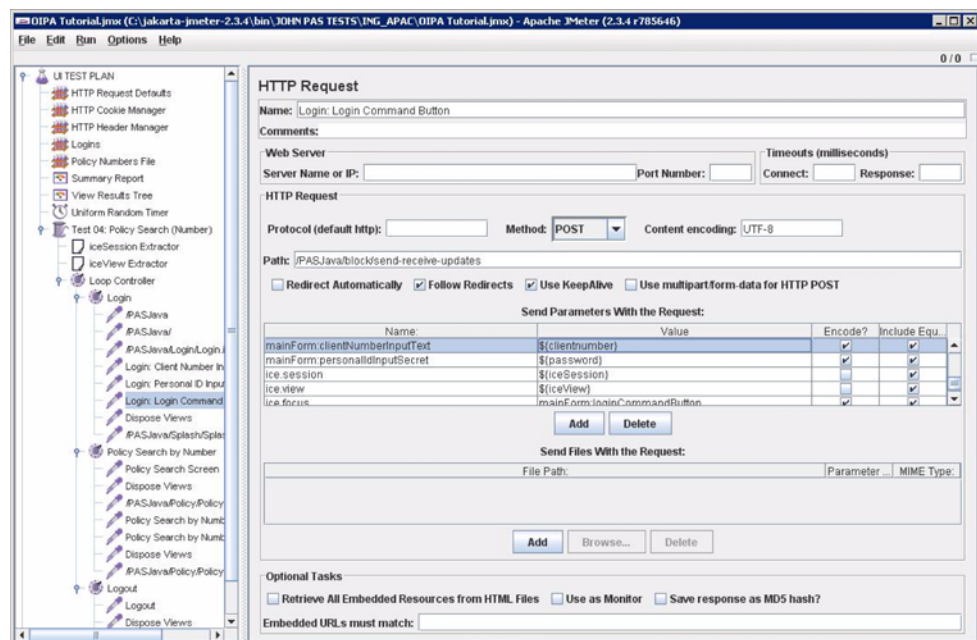


Figure 10: The HTTP Request Section

7. Create an input file, PolicyNumbers.txt, and enter single values, corresponding to valid policy numbers in the application being tested.
8. Add another CSV Data Set Config element corresponding to the policy numbers, as shown in Figure 11.

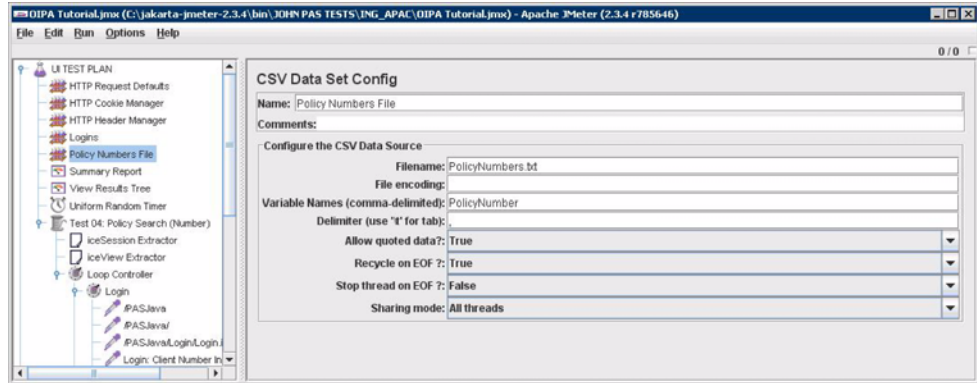


Figure 11: The CSV Data Set Config Element Corresponding to the Policy Numbers

9. Reset the variable for search criteria in the pages where the policy search is executed, specifically, when the find button is clicked, as shown in Figure 12.

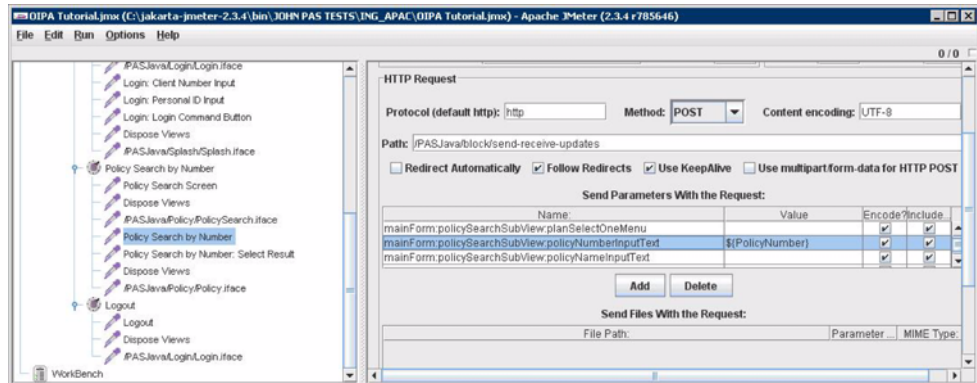


Figure 12: Resetting the Variable for Search Criteria

Extracting ICEfaces Session Variables

In some cases, including the current scenario, special variables will need to be recorded. In this case, the variables are the ice.session and ice.view values that are seen in every page. These values must be dynamically set in each and every page visited.

To configure the ice.session and ice.view values:

1. Right-click **TEST PLAN**.
A menu is displayed.
2. On the menu, click **Add, Post Processors**, and then **Regular Expression Extractor** for ice.session, as shown in Figure 13.

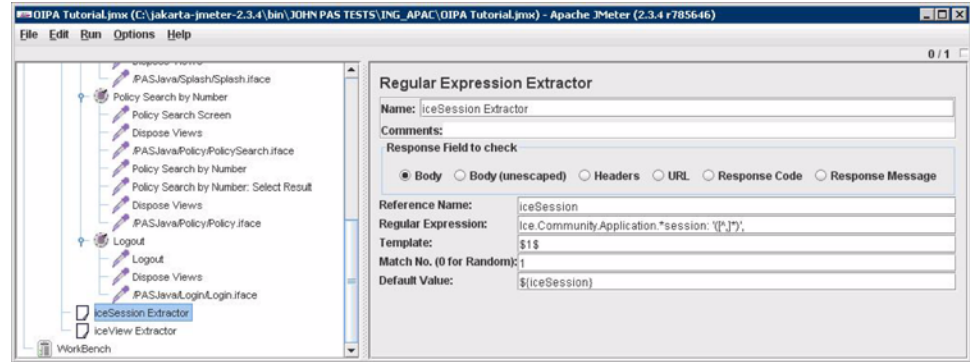


Figure 13: Configuring the `ice.session` Value

3. In the **Regular Expression** field, enter **Ice.Community.Application.*session:'([\^,]*)'**.
4. Right-click **TEST PLAN**.
A menu is displayed.
5. On the menu, click **Add, Post Processors**, and then **Regular Expression Extractor** for `ice.view`, as shown in Figure 14.

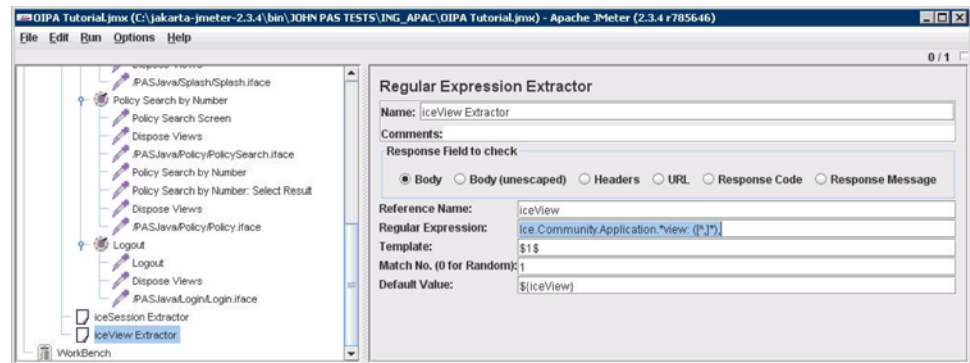


Figure 14: Configuring the `ice.view` Value

6. In the **Regular Expression** field, enter **Ice.Community.Application.*view:'([\^,]*)'**.
7. Open every page, and replace the values for `ice.session` and `ice.view` with `$(iceSession)` and `$(iceView)`, as shown in Figure 15.

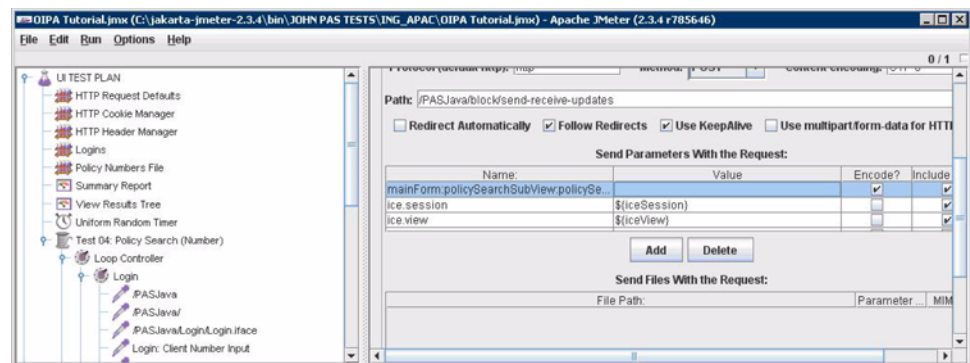


Figure 15: Replacing the Values of the ice.session and ice.view Parameters

- The dispose_views pages are a special case and must be treated differently. In these pages, the ice.session and ice.view data must be added manually and the values parameterized. Every dispose_views page in the test case should look like Figure 16.

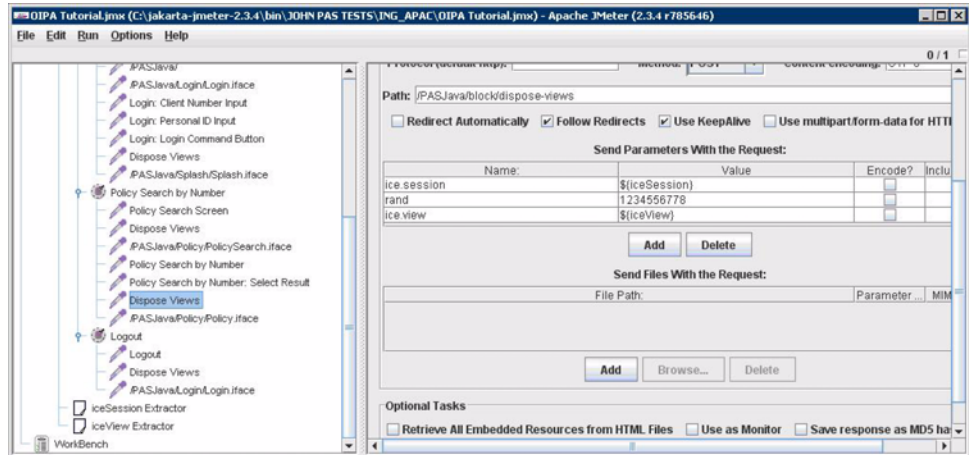


Figure 16: The dispose_views Page

Adding Listeners

Listeners are the report generators of Apache JMeter. To add Listeners to capture results:

- Right-click **TEST PLAN**.
A menu is displayed.
- On the menu, click **Add, Listeners**, and then **Summary Report**.
- If you want to view the results externally, then enter a filename. The file will be generated automatically after results are generated, as shown in Figure 17.

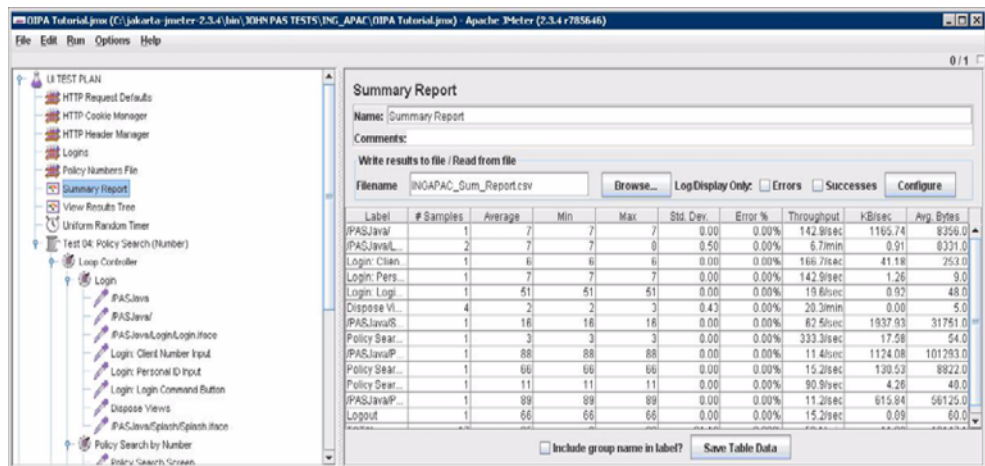


Figure 17: The Summary Report Page

- Right-click **TEST PLAN**.

A menu is displayed.

- On the menu, click **Add**, **Listeners**, and then **View Results Tree**.

The results are generated and the request/response screens can be viewed for every page recorded, so that errors are more visible, as shown in Figure 18.

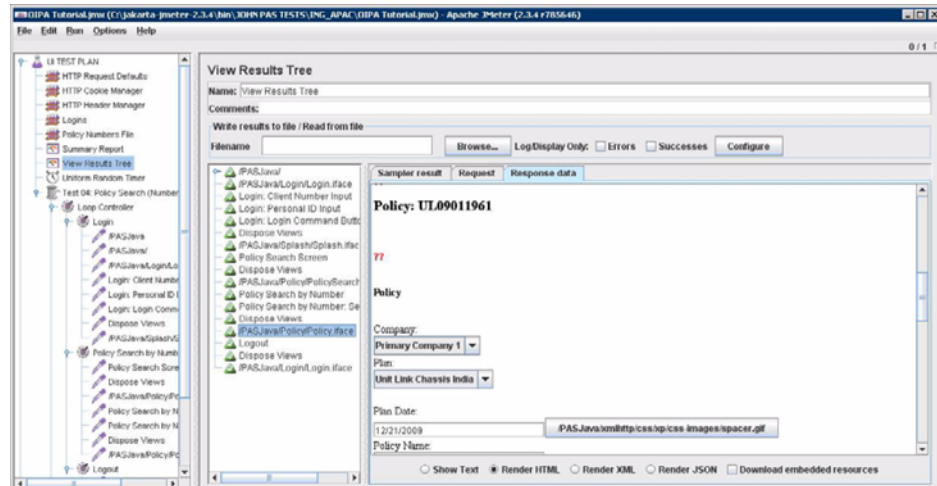


Figure 18: The View Results Tree Page

Adding Timers

You must add timers to record the time taken for the performance. To add timers:

- On the menu, click **Timer**, and then click **Uniform Random Timer**.

The Uniform Random Timer page is displayed.

- Set a realistic time that a user would require to between each page interaction. In this example, set a constant delay of 1000 minutes, with a random delay maximum of 500 minutes, as shown in Figure 19. This means that every user think time will be a random real value between 500 and 1500 ms.

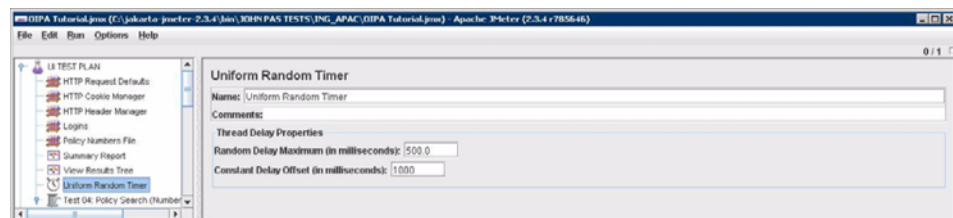


Figure 19: The Uniform Random Timer Page

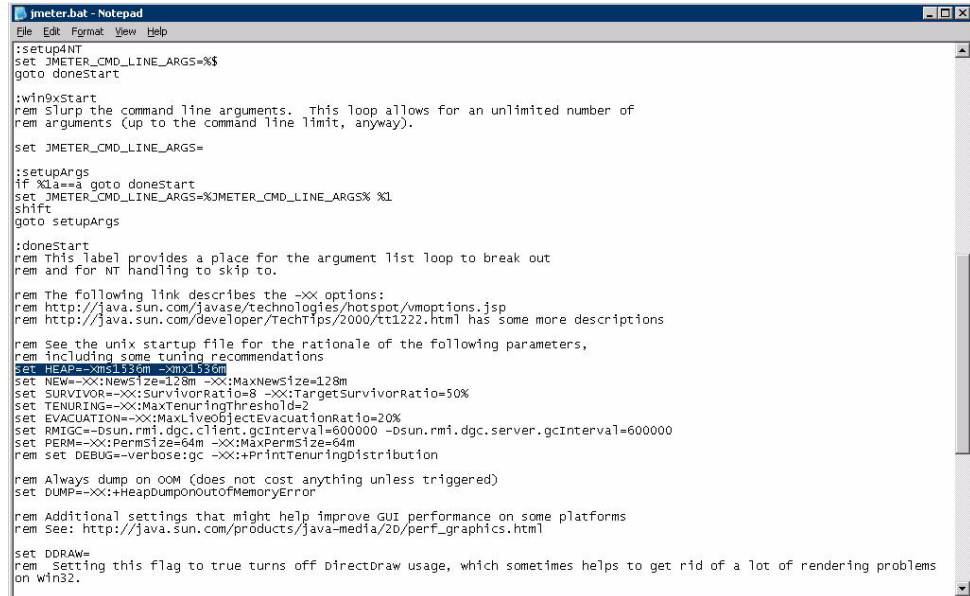
Altering Heap Size

Sometimes if there are many users using Apache JMeter simultaneously, then Apache JMeter may crash. If the system is high end, then you can increase the number of users by altering the heap size.

To change the Java heap size allocation for Apache JMeter:

- Right-click **JMeter.bat**, and then click **Edit**.

2. Change the value to whatever best suits the current environment running Apache JMeter, as shown in Figure 20.



```

jmeter.bat - Notepad
File Edit Format View Help
:setup4NT
set JMETER_CMD_LINE_ARGS=%*
goto doneStart

:win9xStart
rem slurp the command line arguments. This loop allows for an unlimited number of
rem arguments (up to the command line limit, anyway).
set JMETER_CMD_LINE_ARGS=

:setupArgs
if %1a==a goto doneStart
set JMETER_CMD_LINE_ARGS=%JMETER_CMD_LINE_ARGS% %1
shift
goto setupArgs

:doneStart
rem This label provides a place for the argument list loop to break out
rem and for NT handling to skip to.

rem The following link describes the -XX options:
rem http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp
rem http://java.sun.com/developer/techtips/2000/tt1222.html has some more descriptions

rem See the unix startup file for the rationale of the following parameters,
rem including some tuning recommendations
set HEAP=-Xms128m -Xmx128m
set NEW=-XX:NewSize=128m -XX:MaxNewSize=128m
set SURVIVOR=-XX:SurvivorRatio=8 -XX:TargetSurvivorRatio=50%
set TENURING=-XX:MaxTenuringThreshold=2
set EVACUATION=-XX:MaxLiveObjectEvacuationRatio=20%
set RMIGC=-Dsun.rmi.dgc.client.gcInterval=600000 -Dsun.rmi.dgc.server.gcInterval=600000
set PERM=-XX:PermSize=64m -XX:MaxPermSize=64m
rem set DEBUG=-verbose:gc -XX:+PrintTenuringDistribution

rem Always dump on OOM (does not cost anything unless triggered)
set DUMP=-XX:+HeapDumpOnOutOfMemoryError

rem Additional settings that might help improve GUI performance on some platforms
rem See: http://java.sun.com/products/java-media/2D/perf_graphics.html

set DDRAW=
rem Setting this flag to true turns off DirectDraw usage, which sometimes helps to get rid of a lot of rendering problems
rem on win32.

```

Figure 20: The JMeter.bat File

Defining Scope

Scope is very important, as different elements of the test scenario would affect different levels of the test execution.

The following are some ways of defining scope:

- All of the Configuration Elements, Listeners, Timers, and CSV Data Sets can be placed at the top level, outside of the Thread Group, in case other Thread Groups need to be added in the future for other test cases. One configuration manager, listener, CSV Data Set, and Timer can encompass all test cases.
- Regular Expression Extractors must be placed inside of each Thread Group, as the `ice.session` and `ice.view` variables need to be different for each Thread Group.

Figure 21 shows the final result of the test case.

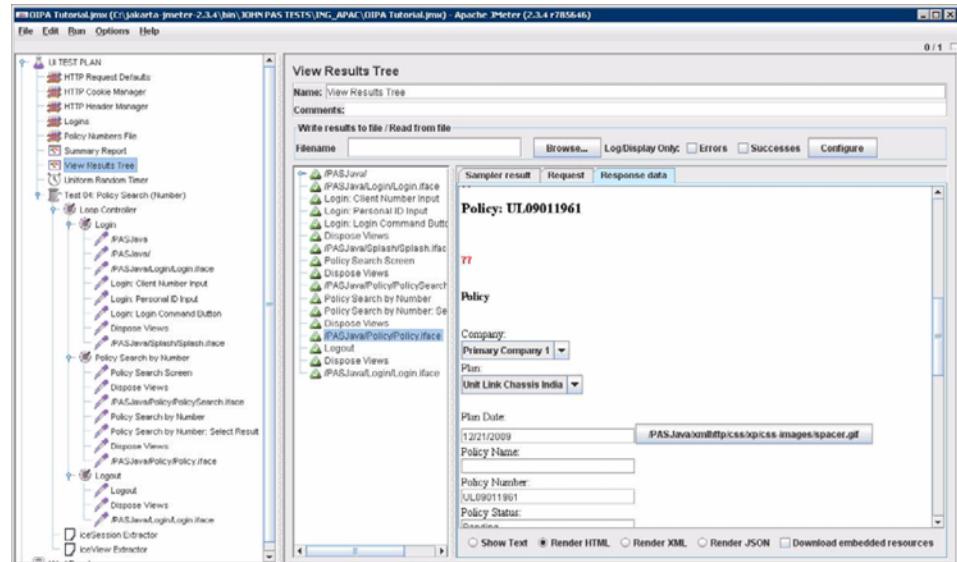


Figure 21: The Final Result of Performance Testing the OIPA by Using Apache JMeter

LOAD TESTING OIPA USING NEOLOAD

This section describes the benefits of using NeoLoad for performance testing and the steps involved in performance testing OIPA using NeoLoad.

This section includes the following topics:

- *Introduction to NeoLoad*
- *Benefits of Using NeoLoad*
- *Load Testing OIPA Using NeoLoad*

Introduction to NeoLoad

NeoLoad is a load testing software. NeoLoad helps in testing Web applications, simulating users in a realistic way, and in analyzing the behavior of the server. The technology used in NeoLoad allows users to test more quickly, efficiently and repeatedly. Using NeoLoad you can deploy your applications to any architecture they may use, such GWT and AJAX Push.

Benefits of Using NeoLoad

The following are some of the benefits of using NeoLoad for performance testing:

- Recording and testing of scripts is very simple, without affecting scalability and test coverage.
- Generating professional looking reports and graphs is quick and easy.
- Automatic handling of application parameters (ICEfaces) and report generation provides great savings in man-hours during the load testing process.

- Reports include smart pinpointing of the application's critical performance issues, including highest response time pages.

Load Testing OIPA Using NeoLoad

This section describes performance testing OIP using NeoLoad. This section includes the following topics:

- *Creating a New Project*
- *Recording Performance Details*
- *Post-Recording Wizard*
- *Designing Performance Testing*
- *Excluding Non-Essential Pattern*
- *Checking for Errors*
- *Extracting Variables*
- *Creating Data Set Input Files*
- *Designing Populations Tab*
- *Creating Runtime Parameters*
- *Executing the Test*
- *Viewing Results and Generating Reports*
- *Creating a Standard Report*
- *Creating a Comparison Report*

Creating a New Project

You must create a new NeoLoad project to start performance testing any system using NeoLoad. To create a new project:

1. On the Welcome Screen, click **New Project**.
The New dialog is displayed.
2. In the **Project name** field, enter a name for the project and in the **Directory** field, enter a valid directory to save the project in, as shown in Figure 22.

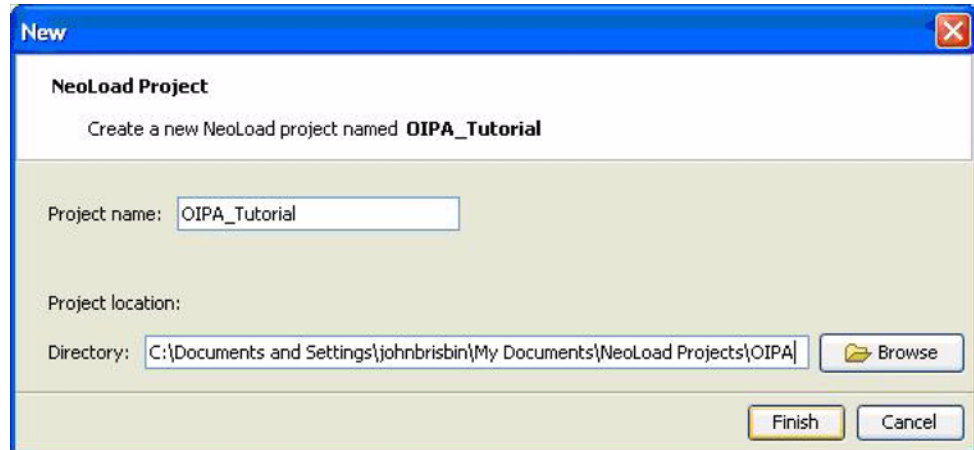


Figure 22: The New Dialog

3. Click **Finish**.

The NeoLoad window is displayed, as shown in Figure 23.

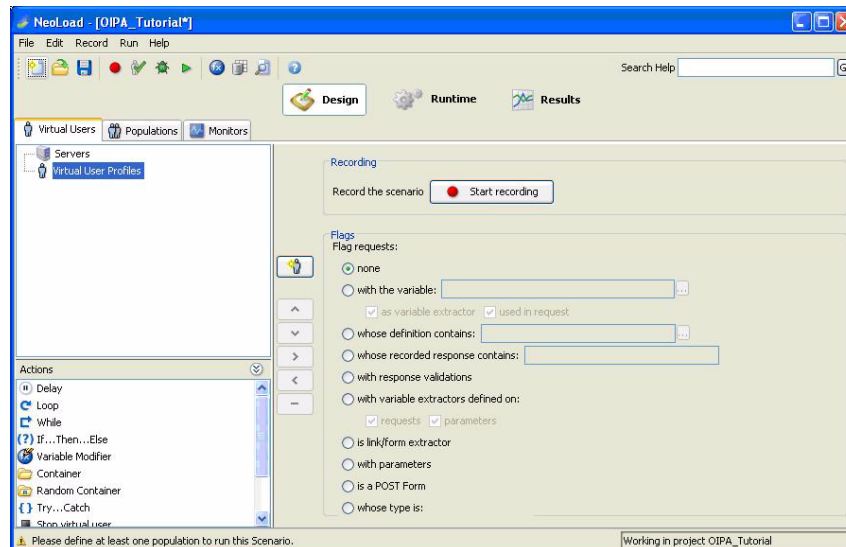


Figure 23: The NeoLoad Window

4. Click **Start recording**.

The Recording dialog is displayed.

Recording Performance Details

To record the performance details:

1. In the **Name** field of the Recording dialog, enter a name for the virtual user, as shown in Figure 24.



Figure 24: The Recording Dialog

2. Click **OK**.

The NeoLoad - Recording of Virtual User dialog is displayed, as shown in Figure 25. Simultaneously, the preferred browser, which is Internet Explorer version 7 is also displayed.

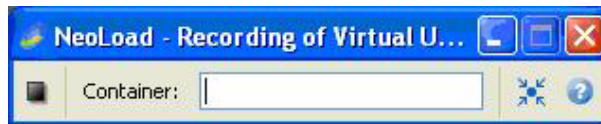


Figure 25: The NeoLoad - Recording of Virtual User Dialog

The NeoLoad recording window handles Container names. Container names are used to parse the script into manageable blocks (known also as 'Transactions' in LoadRunner or 'Controllers' in JMeter). Each new page opened or important action taken should be given a descriptive container name, which is crucial in report analysis. For example, log in to NeoLoad, click policysearch, policyload, then policysave, and then finally log out of NeoLoad.

3. In the **Container** field of the NeoLoad - Recording of Virtual User dialog, enter Login.
4. In the address bar of Inter Explorer, enter the URL and begin scripting, changing container names when necessary.
5. After you have logged out of the application, click the **Stop** button on the NeoLoad recording window to end the recording.

Post-Recording Wizard

The Post-recording window shown in Figure 26 appears after you have completed the recording.

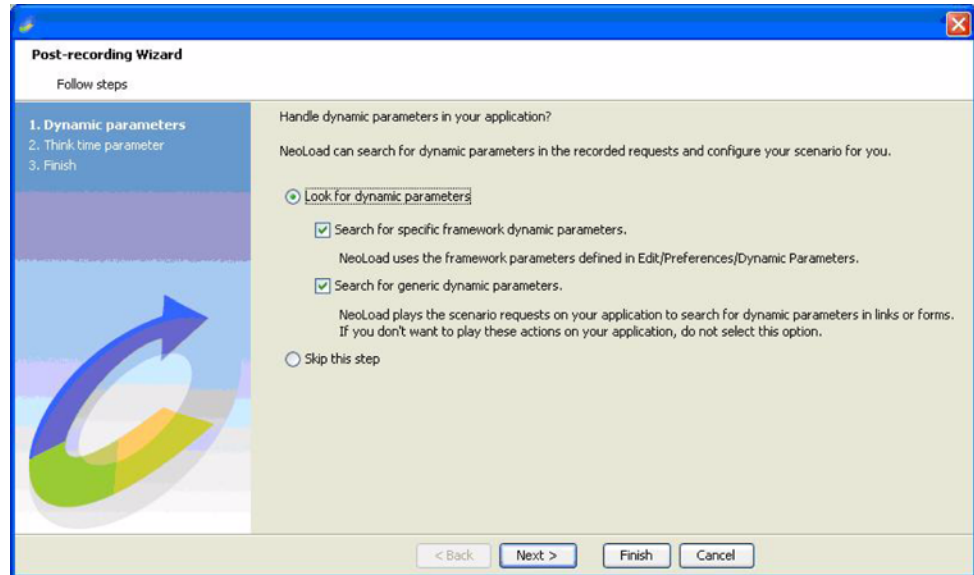


Figure 26: The Post-recording Wizard

To handle dynamic parameters in the application:

1. Under Look for dynamic parameters, select both the options.
2. Click **Next**.

This will automatically find and correlate the ICEfaces session and view variables, which is absolutely crucial in the OIPA application.

3. NeoLoad will report having found ICEfaces dynamic parameters. Select Apply the Changes, and then click Next twice.
4. Overwrite the recorded think times with a desired amount in minutes. This value can be changed at any time in the main screen.

The main NeoLoad design window with the Virtual User script you entered is displayed.

Designing Performance Testing

In the main NeoLoad design window, you can override the think time, and add a random delay percentage. For example, you can add 1500 ms +/- 50% to any random real time from 1-2 seconds, as shown in Figure 27.

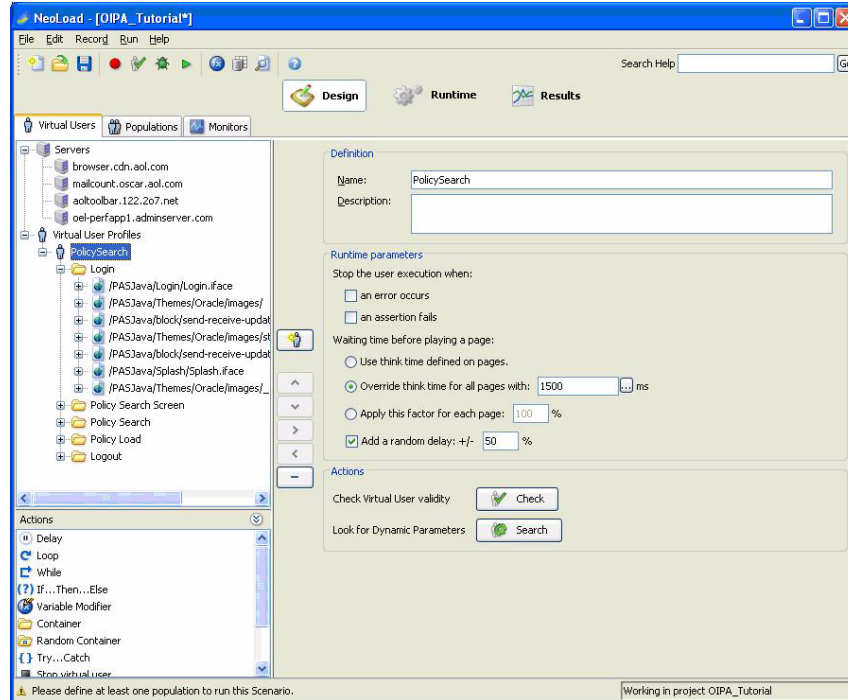


Figure 27: The NeoLoad Design Window

Excluding Non-Essential Pattern

Certain pages must be omitted from the script recording, including locations of embedded images, as they can cause NeoLoad to throw errors during testing. To exclude non-essential patterns:

1. On the menu, click **Record, Recording Preferences**.

The Preferences dialog is displayed.

2. Under the General setting tab, select **HTTP Recorder**.
3. In the **Patterns to exclude** field, enter any non-essential page patterns. For example: `http://.*/Themes/* & http://.*/Scripts/*`, as shown in Figure 28.

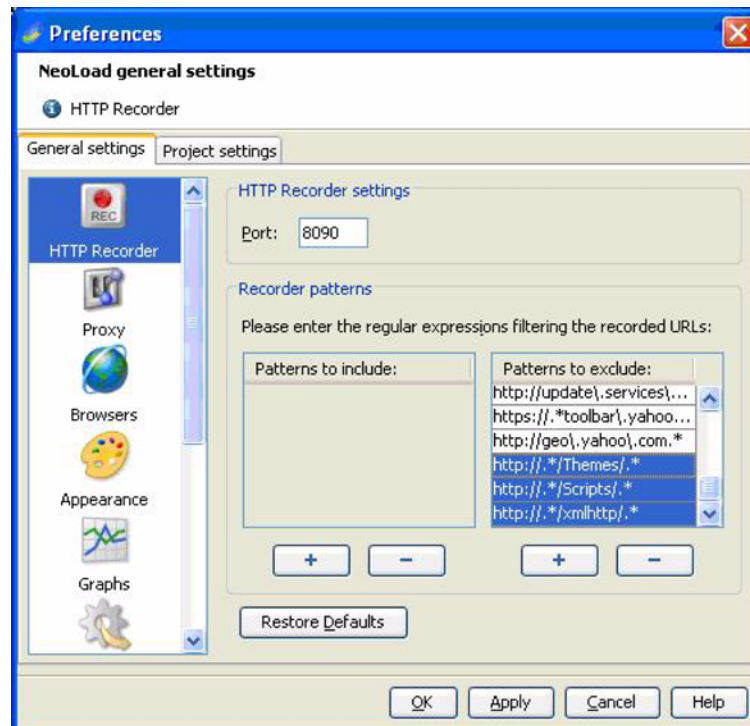


Figure 28: The Preferences Dialog

Checking for Errors

To prove the validity of the script, and handle any errors

1. Click **Check**.

The Check Virtual User window is displayed, as shown in Figure 29.

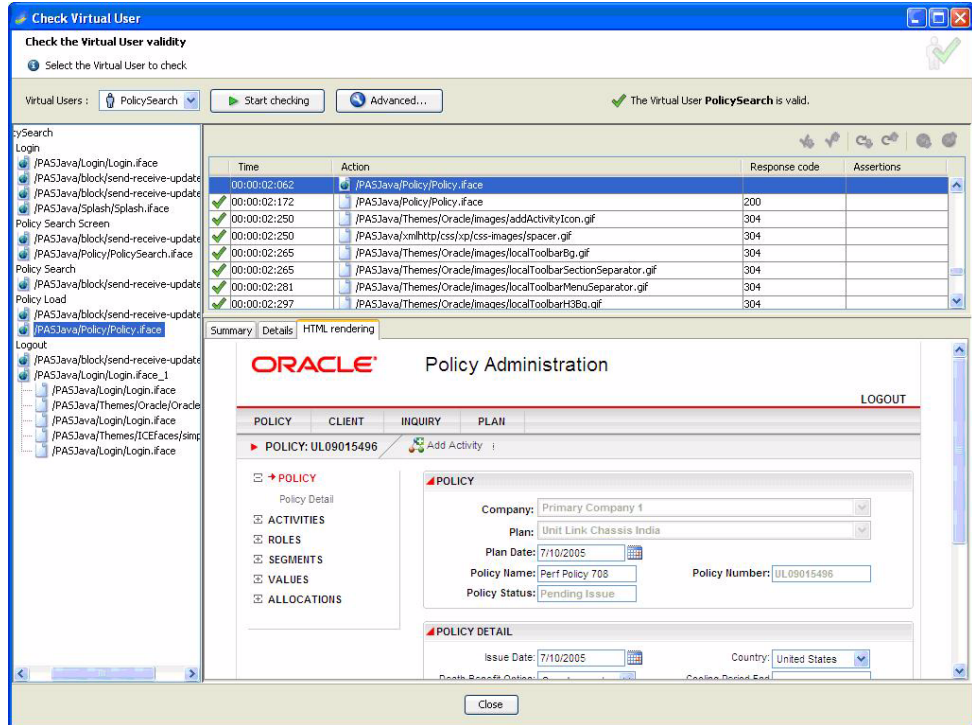


Figure 29: The Check Virtual User Window

2. Click **Start checking** to run a single user test.

If there are any errors, they will be reported here. Also, by clicking on any page, the HTML can be rendered in order to check that the screen loads properly and that the correct information is present.

Note This step must always be performed prior to extensive testing.

Extracting Variables

Sometimes, values must be extracted from the server response to be applied to other areas of the test. An example of this is creating an OIPA policy, extracting the generated policy number, and using it in a search operation.

To extract variables:

1. In the NeoLoad window, click the **Policy Save** page, and then click **Advanced**, as shown in Figure 30.

The Advanced dialog is displayed.

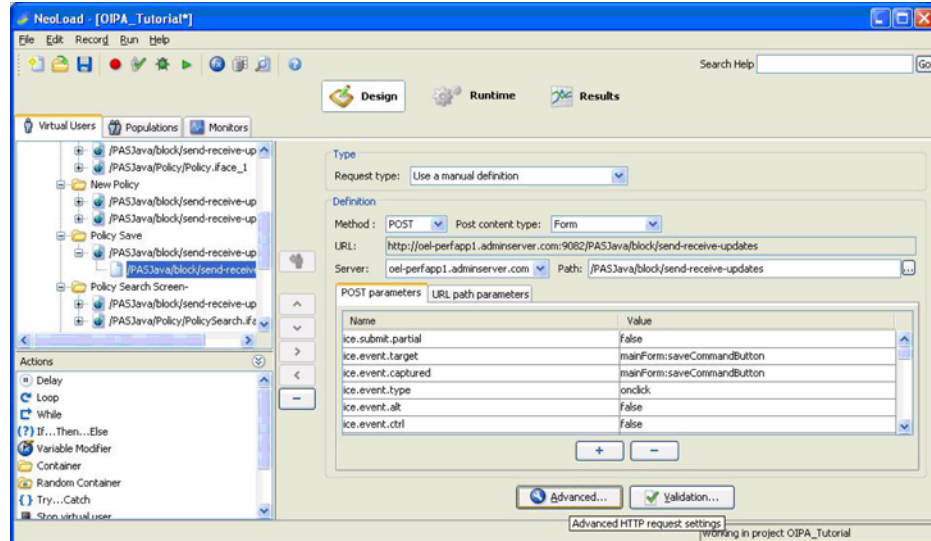


Figure 30: The NeoLoad Window

2. Click the **Variable Extractors** tab and then click +, as shown in Figure 31.

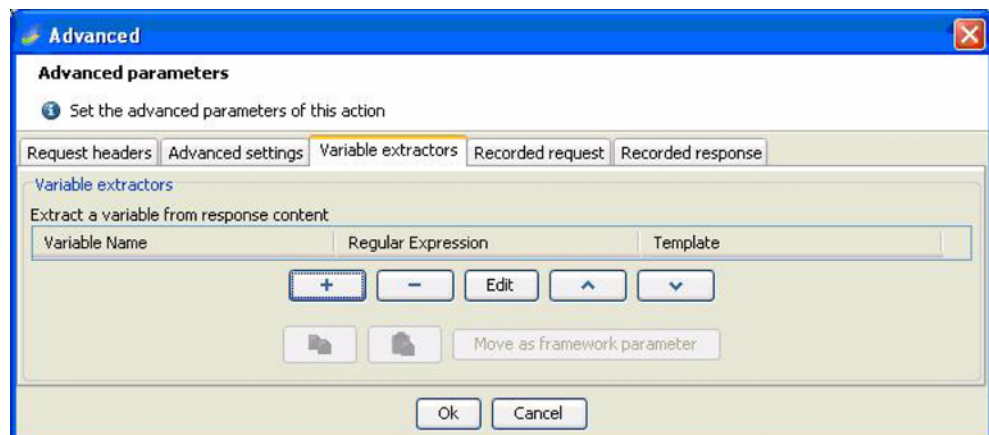


Figure 31: The Advanced Dialog

The Variable Extractor dialog is displayed. The variable extractors are defined by regular expressions in this dialog.

3. Enter the left and right bound information based on the context of the desired value in the response, as shown in Figure 32.

If the value is extracted correctly, then the value is validated at the bottom of the screen.

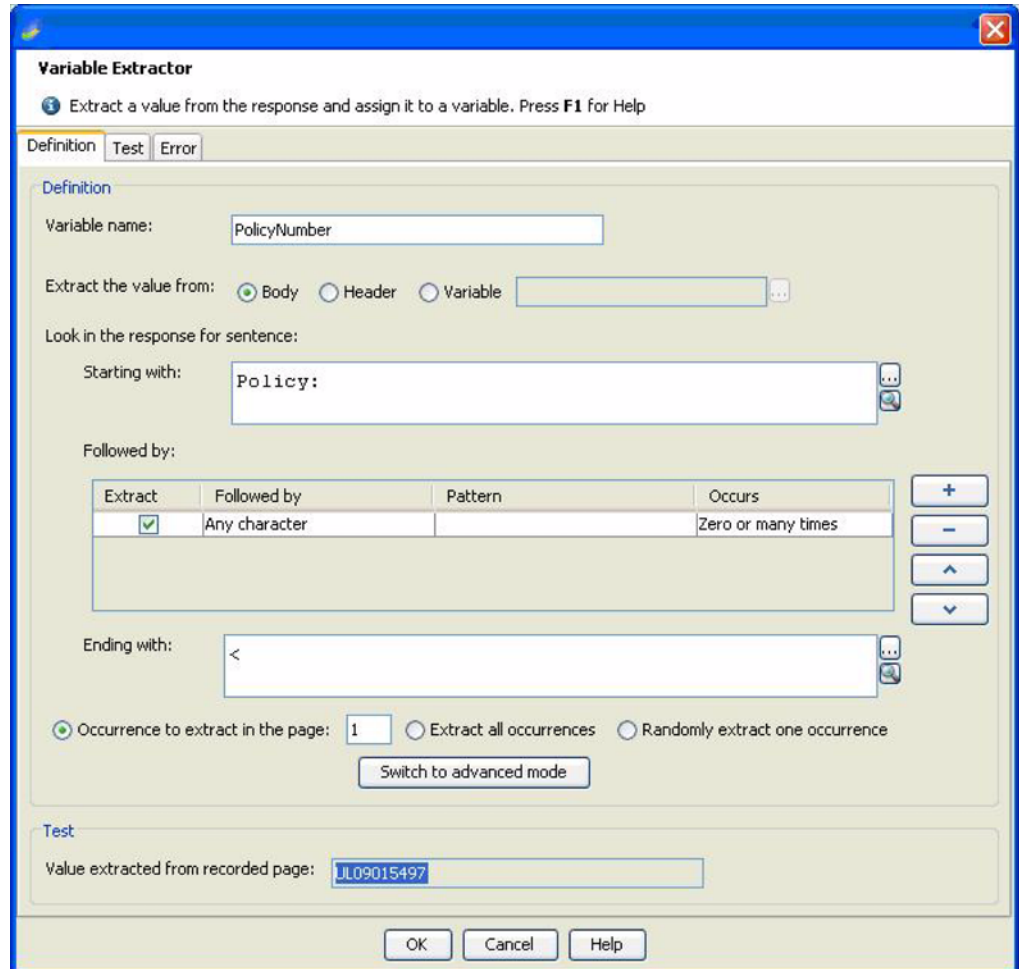


Figure 32: The Variable Extractor Dialog

After the policy number is correctly extracted, apply this value to the policy search screen.

4. Open the Policy Search page, and replace the policynumberinputtext field value with `${PolicyNumber}`, as shown in Figure 33.

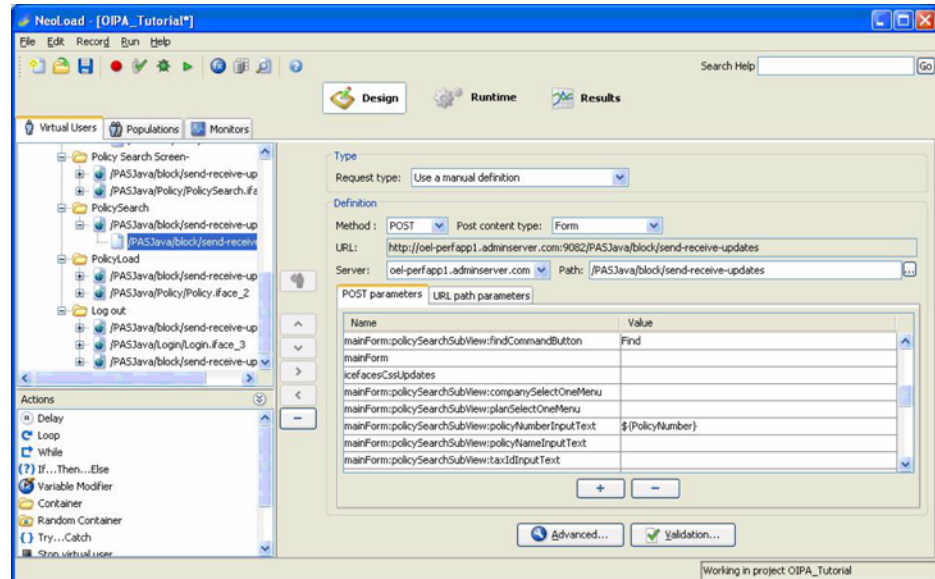


Figure 33: Changing the Policy Number in the Policy Search Page

Now, the test creates a policy and a search for the policy that was just created.

Creating Data Set Input Files

Often, tests require several values to be used, which will have to be passed into the script through comma separated variable (CSV) files. To create data set input files:

1. Create a file named PolicyNumber.csv with the following information, as shown in Figure 34.

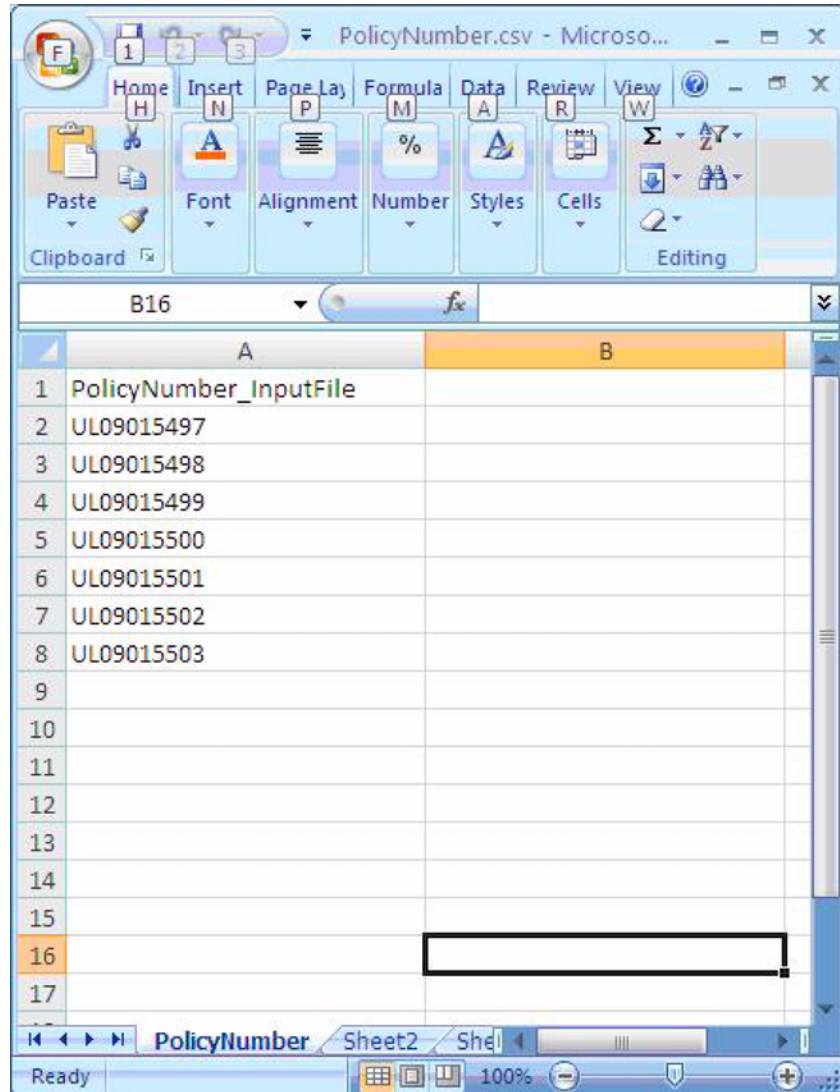


Figure 34: The PolicyNumber.csv File

2. Replace policy numbers with valid numbers in the system.
3. In the NeoLoad window, click **Edit**, and then click **Variables**.
4. Click **New Variable**, and then select **File** from the list.
5. Navigate to the file you just created, and select it, as shown in Figure 35.

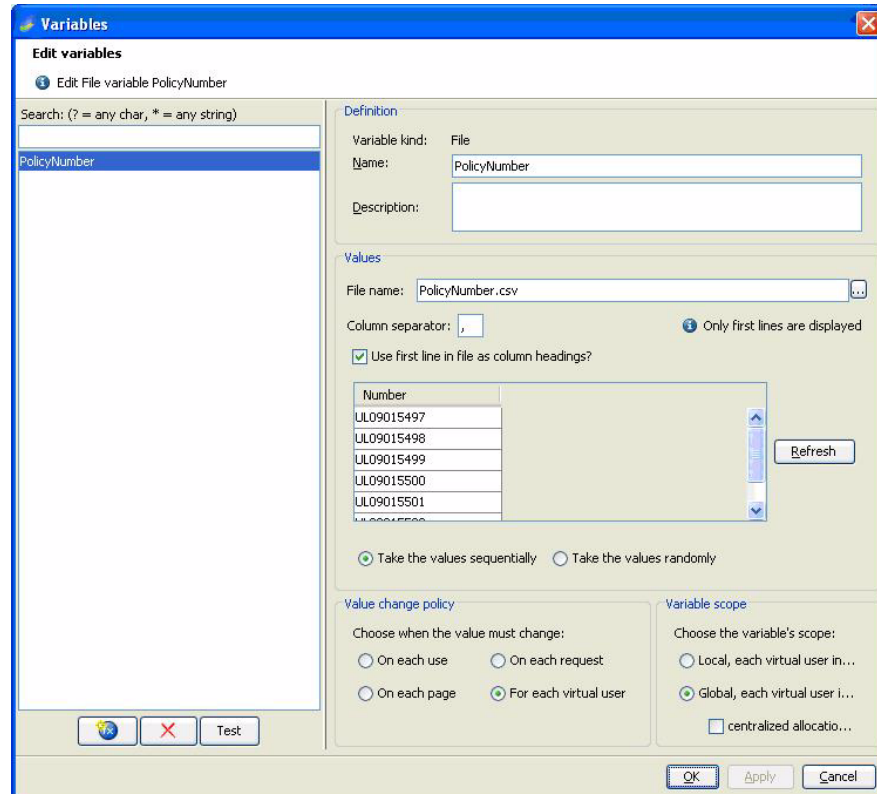


Figure 35: The Variables Dialog

6. In the **Column separator** field, enter ,.
7. Select **Use first line in the field as column heading?**.
8. Under Value Change Policy, select **For each virtual user**.
9. Click **OK**.
10. Navigate to the Policy Search screen, and replace the policynumberinputtext value with a variable `${filename.columnname} --- ${PolicyNumber.Number}`, as shown in Figure 36.

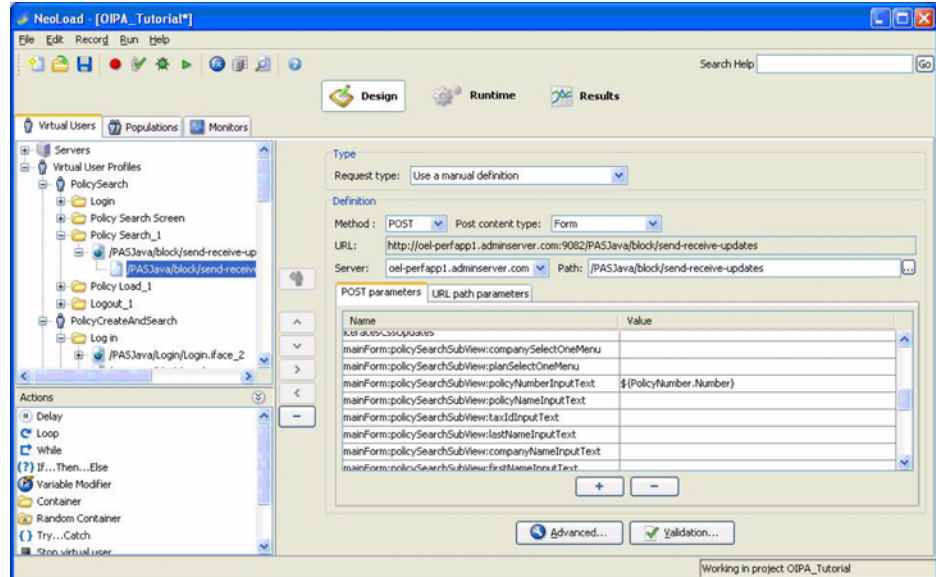


Figure 36: The Policy Search Screen

The policy number used in the search will be pulled in to the script from the CSV file during execution.

Designing Populations Tab

To design populations tab:

1. In the NeoLoad window, click the **Populations** tab to create a population.
2. In the **Name** field, enter descriptive name. In this example, enter PolicySearch_Users, as shown in Figure 37.

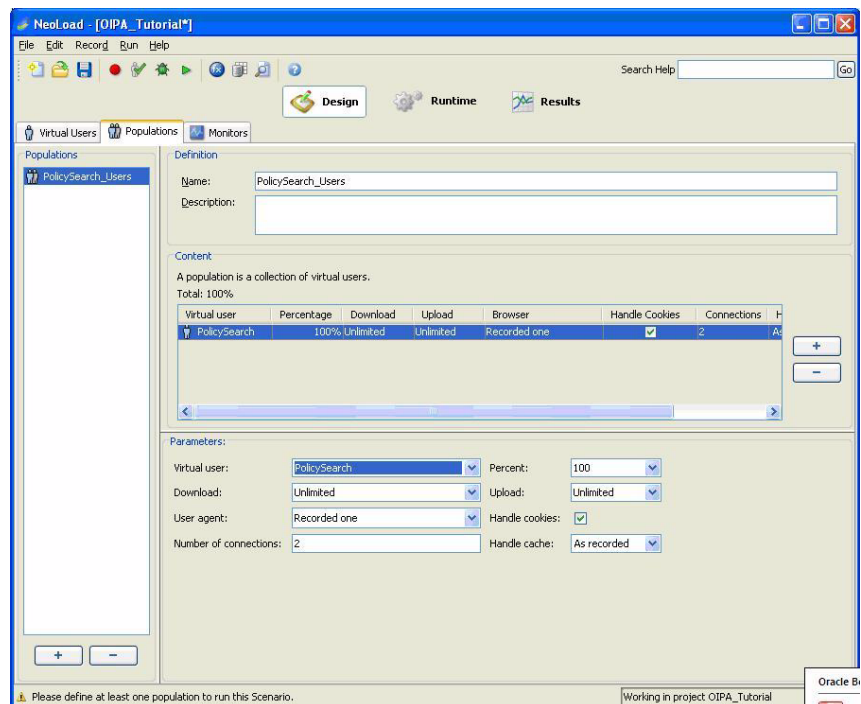


Figure 37: Creating a Population

On this screen, the population can be set to the PolicySearch test or any other test, if more than one is scripted. If two or more populations are created, then the Percent option sets their overall percentage. Generally, these will be set evenly, such as 1=100%, 2=50%, 3=33%, and 4=25%.

Creating Runtime Parameters

To create the runtime parameters:

1. When the Design phase is completed, on the NeoLoad window, click Runtime. The Runtime screen is displayed, as shown in Figure 38.

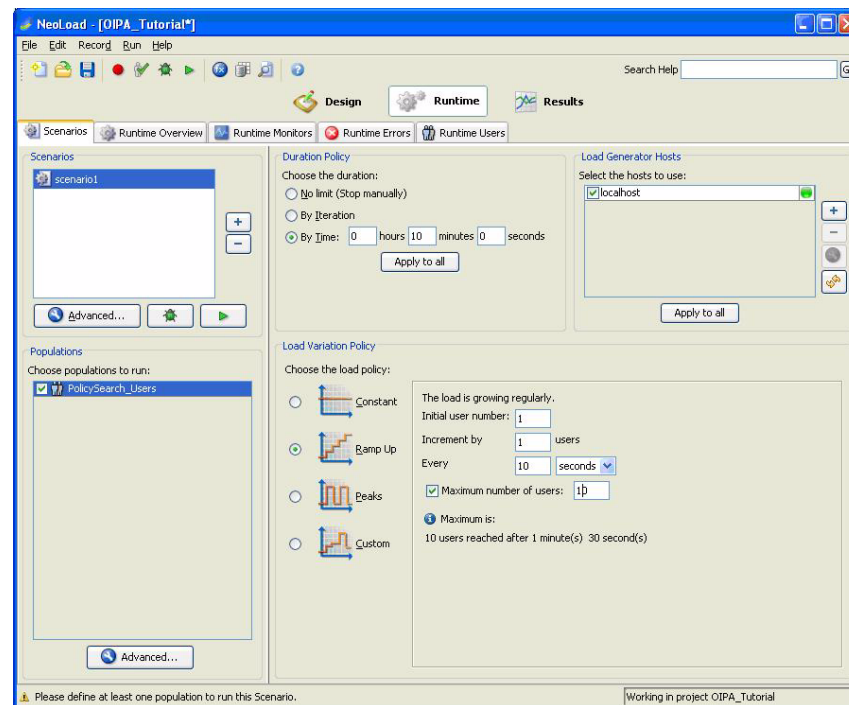


Figure 38: The Runtime Screen

This Runtime screen handles all of the runtime parameters.

- Duration: duration can be handled by time, by number of script iterations, or can be set to run continuously until the user intervenes.
 - Load policy: load policy defines how the virtual users are added to the test. Constant or Ramp Up are the most commonly used, but custom policies can be defined if desired.
 - User count: here, the number of users is defined. Any number of users can be included, license permitting.
2. Click the green Play button to begin running the test.
 3. Enter a descriptive name for the test. For example, IPA_10Users_10Minutes_Datetime.

Executing the Test

The following screens can be monitored live during test execution:

- **Runtime Overview:** includes real-time graphs and general runtime statistics, as shown in Figure 39.

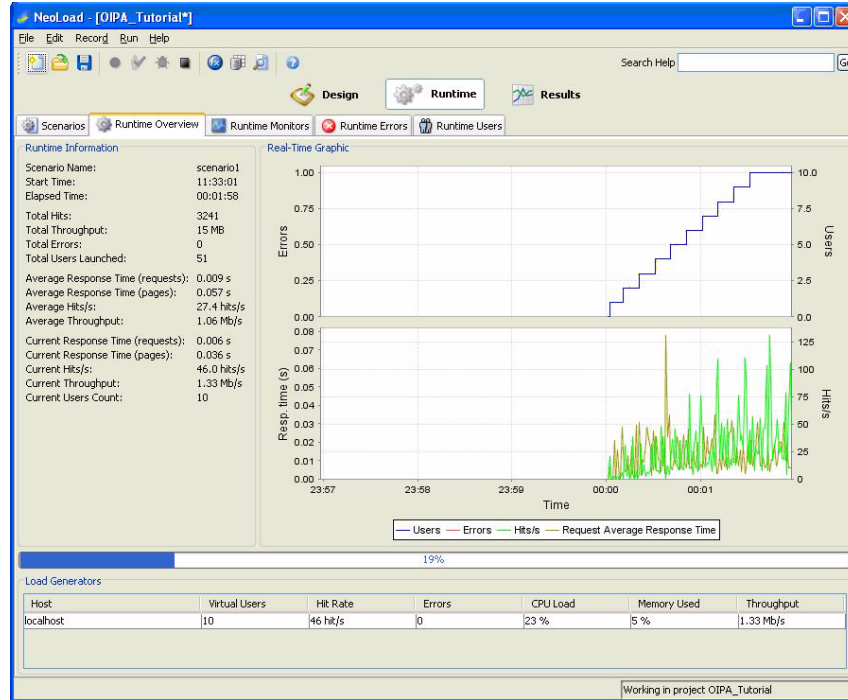


Figure 39: The Runtime Statistics of the Test

- **Runtime Monitors:** more detailed graphs, including those of included load generators, as shown in Figure 40.

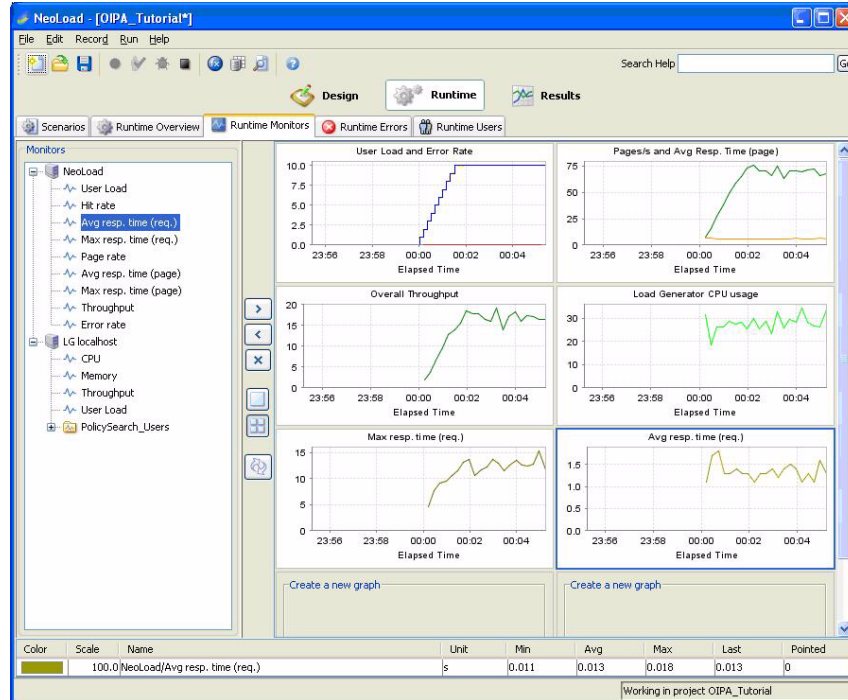


Figure 40: The Runtime Monitors

- Runtime Errors: displays errors (when applicable)
- Runtime Users: follows script execution per virtual user, as shown in Figure 41.

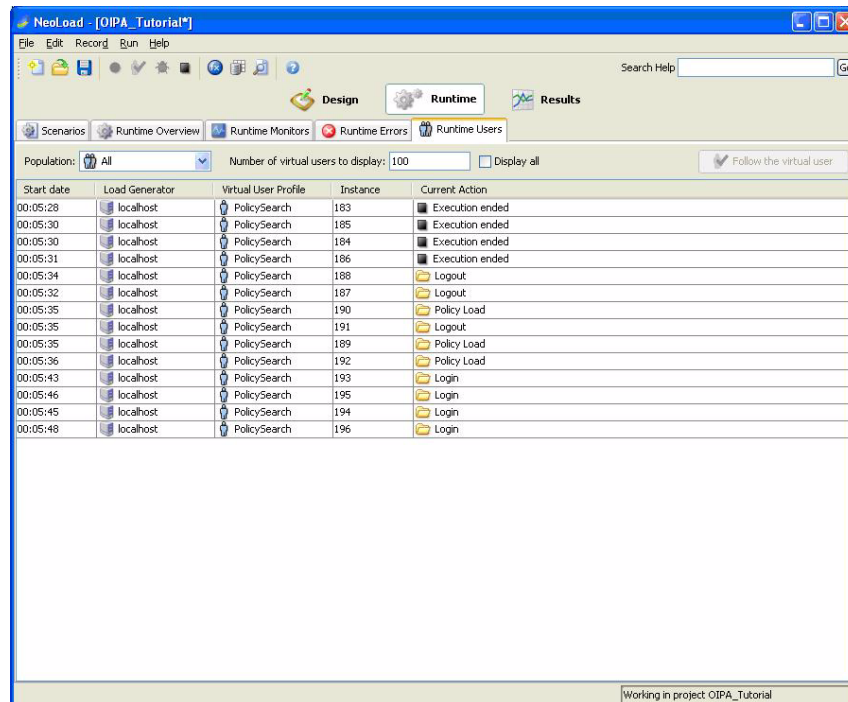


Figure 41: The Runtime Users

Viewing Results and Generating Reports

After test execution, the Results page opens automatically, and the following results summary is given:

Values, Graphs, and Errors can also be viewed if desired, as shown in Figure 42.

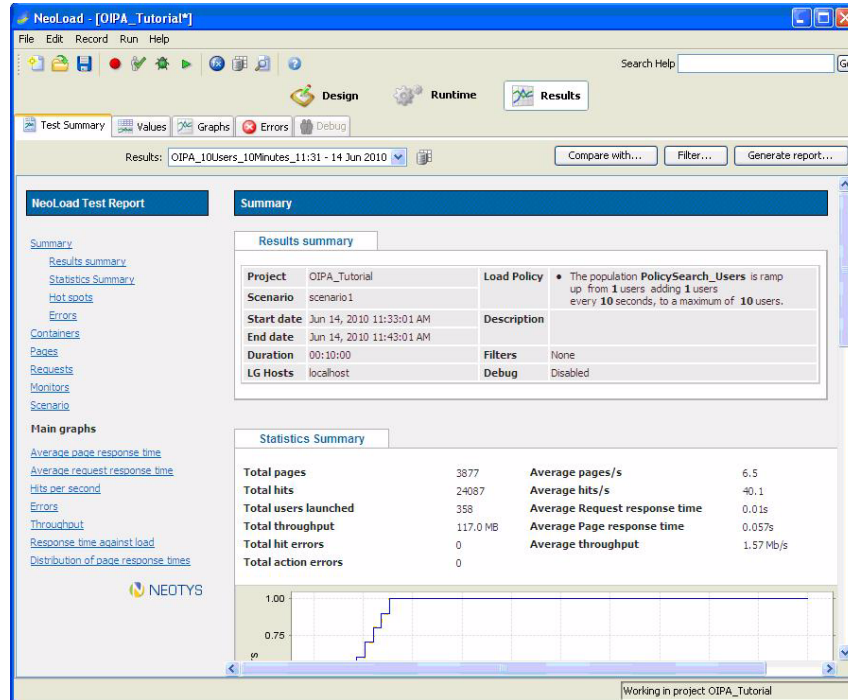


Figure 42: The Result Summary of the Test

To view graphs and add them to the report:

1. Highlight a predefined template and then click the > symbol to add it to the list of graphs, as shown in Figure 43.

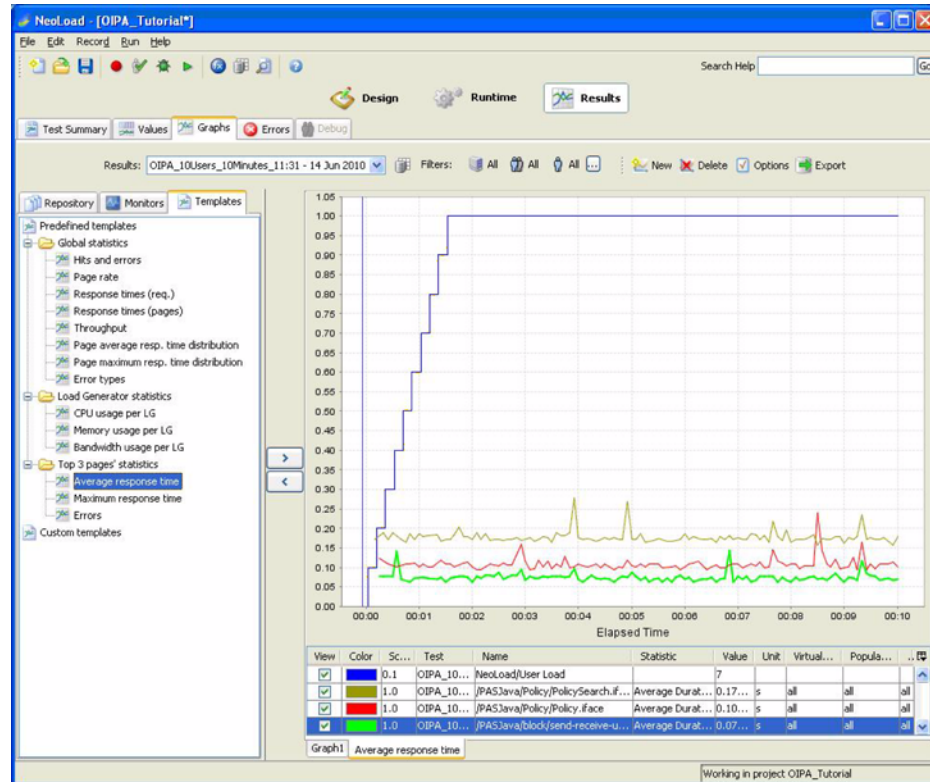


Figure 43: Graphical Result of the Test

2. On the Test Summary screen, click **Generate Report** to create the report.
The Generate Report dialog is displayed.

Creating a Standard Report

To create a standard report:

1. In the Generate Report dialog, select **Standard Report**.
This generates a common single-test report.
2. Click **Next**.
The Test Result Report dialog is displayed.
3. Select the output format, the content to be included, and the output folder, as shown in Figure 44.

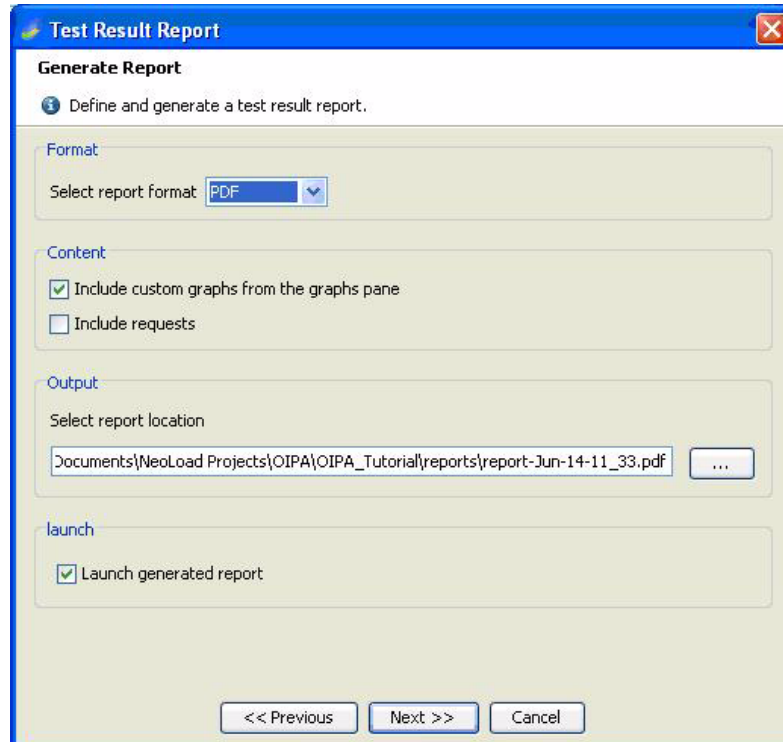


Figure 44: The Test Result Report Dialog

4. Select Include custom graph from the graphs pane to include or exclude any included graphs.
5. Click **Next**.

The Test Result Report dialog is displayed, as shown in Figure 45.

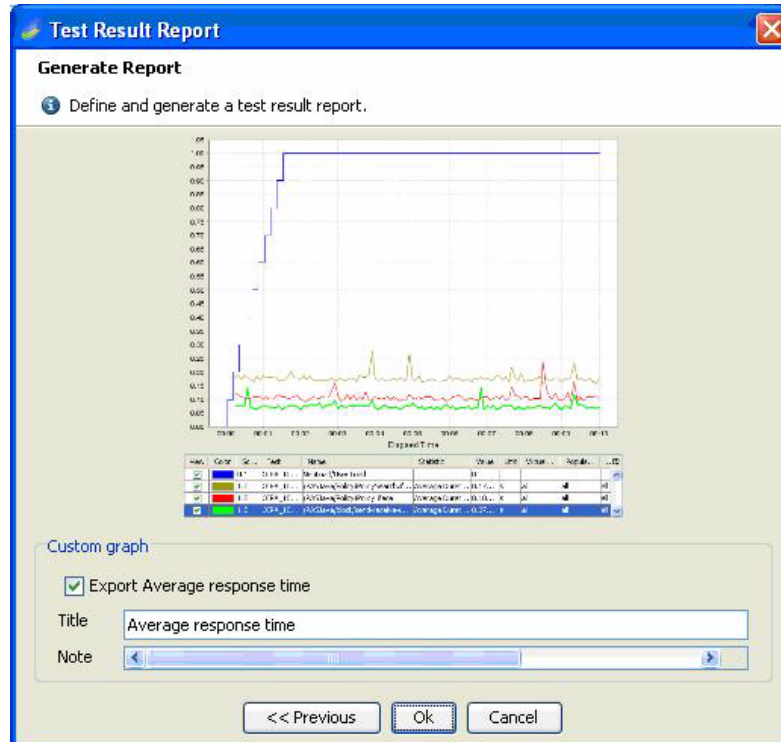


Figure 45: The Test Result Report Dialog

6. Click **OK**.

The report is saved and displayed in the specified format, as shown in Figure 46.

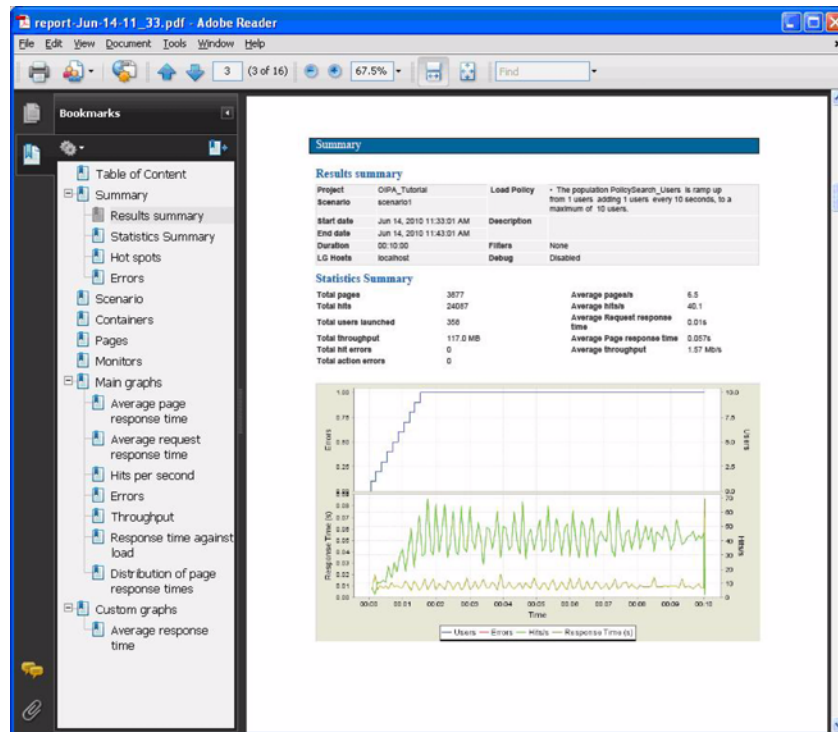


Figure 46: Summary of the Test result

Creating a Comparison Report

A Comparison report compares two different test executions. To create a Comparison report:

1. In the Generate Report dialog, select **Comparison Report**.

The comparison report is generated with graphs. This is an example of comparing a 10 and 60 user report. The 60 user response times are much higher than the 10 user response times, which is to be expected.

2. Click **Next**.

A Comparison report is displayed, as shown in Figure 47 is displayed.

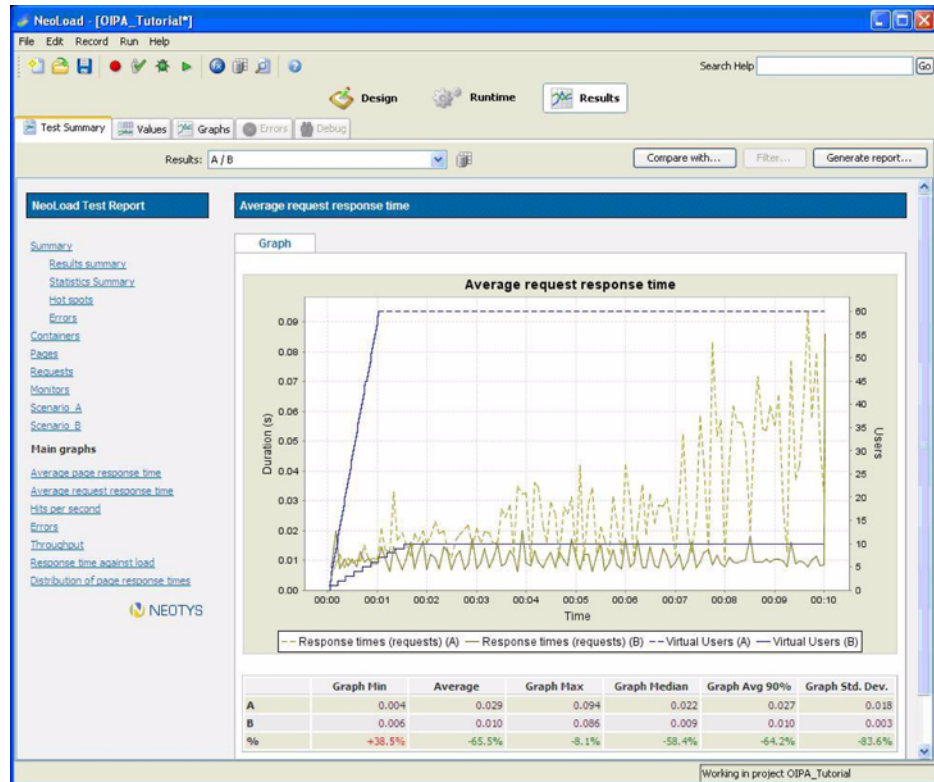


Figure 47: The Comparison Report