Oracle Insurance Policy Administration Load Testing Methods

Letter-Style Template

Release 9.3

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
- Documentation Accessibility

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

ORACLE®

- 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

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 that 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
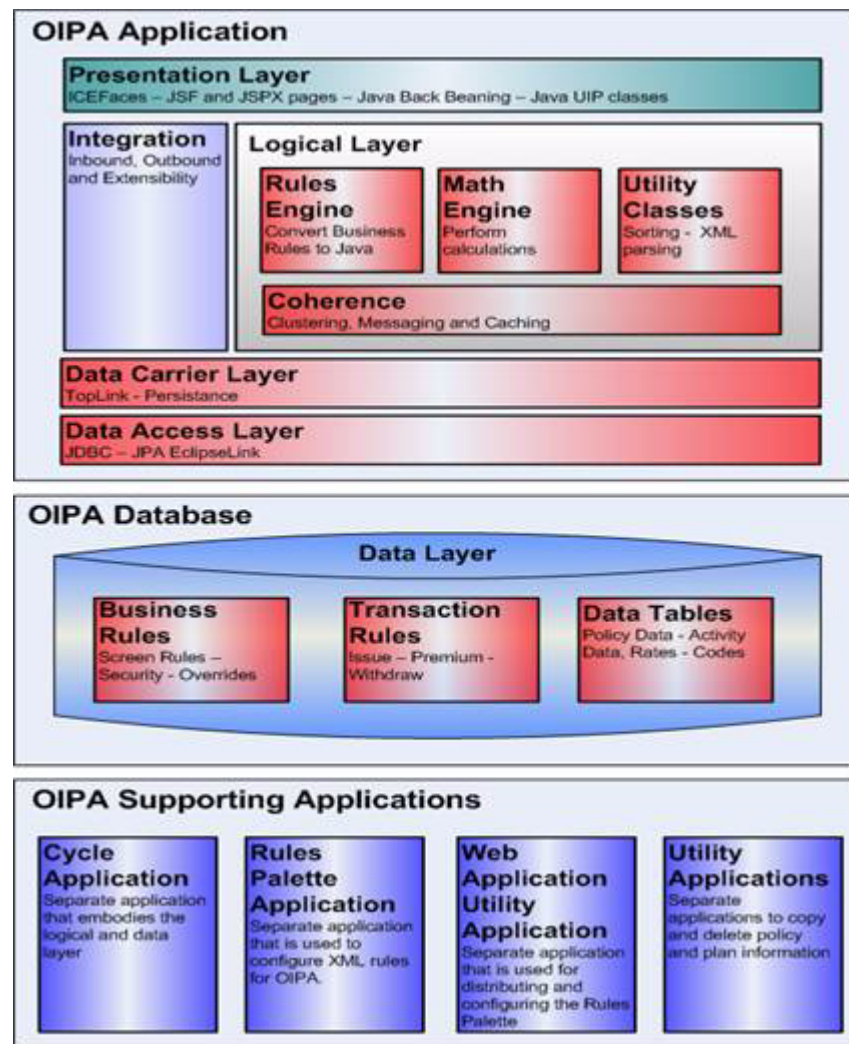
This figure shows the OIPA architecture.

****

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 retrieve data 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.
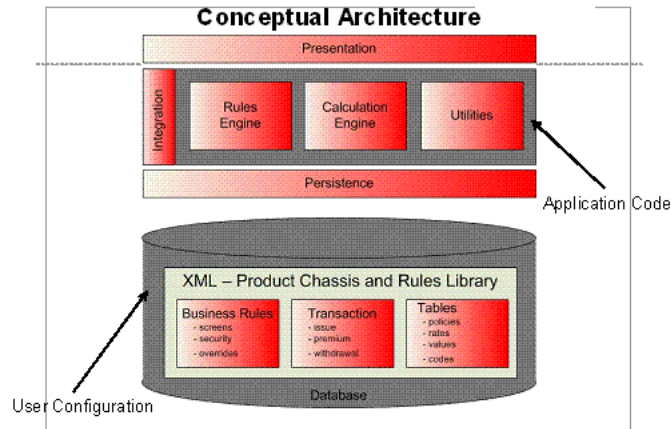
■ Persistence

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 build out 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 the OIPA

This figure shows the conceptual architecture of the OIPA. The conceptual architecture of OIPA includes the following layers: Presentation, Integration, Persistence, and the Database layer.

****

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
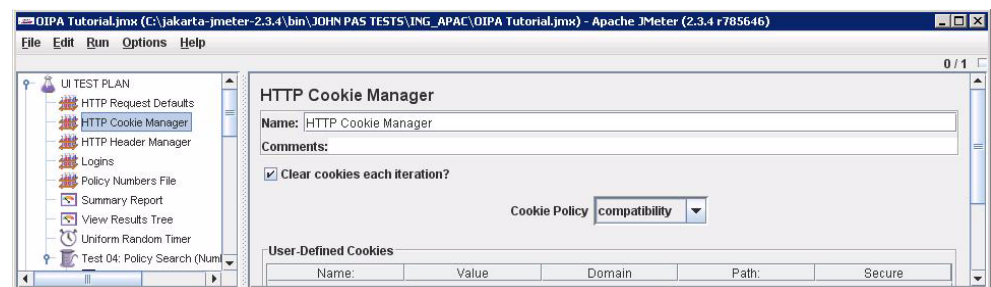
This figure shows the HTTP Cookie Manager section.

****

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
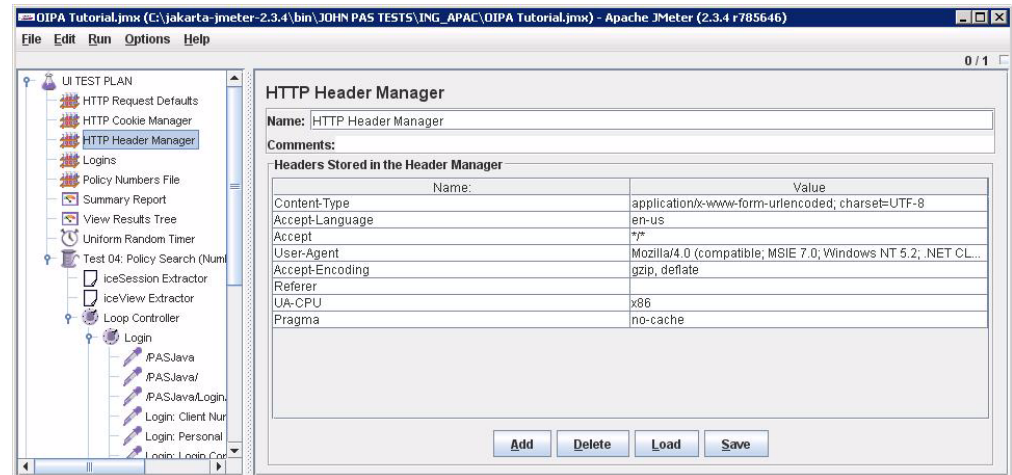
This figure shows the HTTP Header Manager section.

****

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
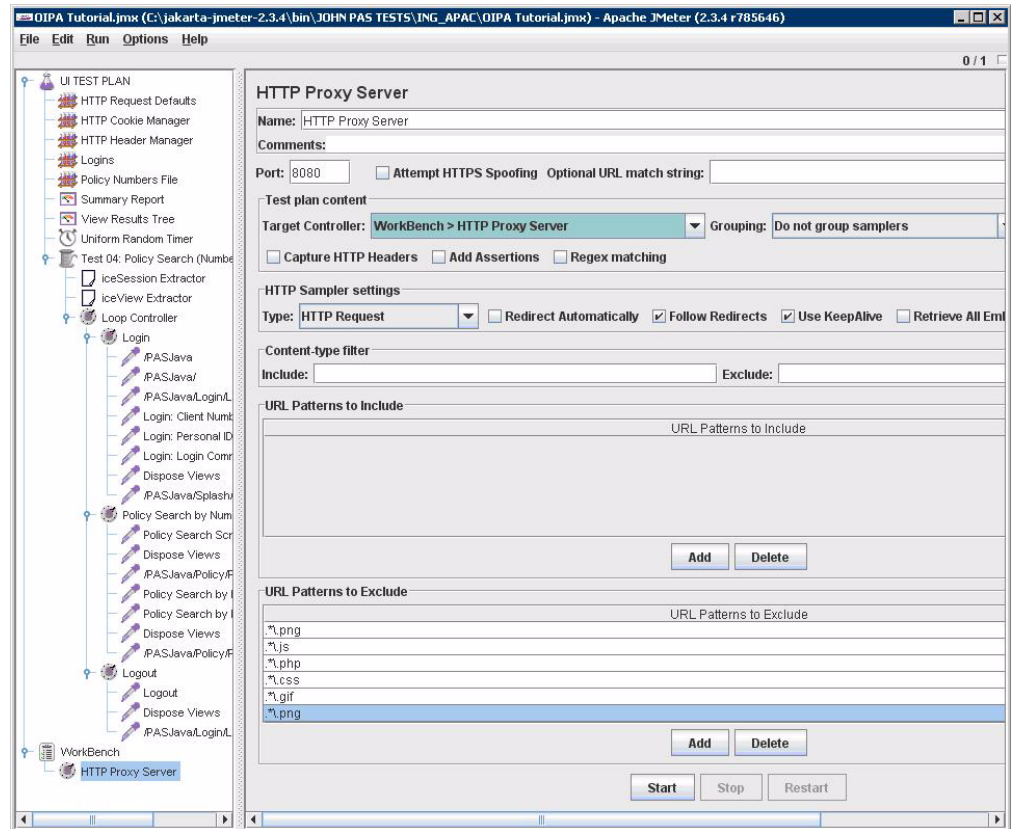
This figure shows the HTTP Proxy Server section.

****

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   Local Area Network (LAN) Settings Dialog
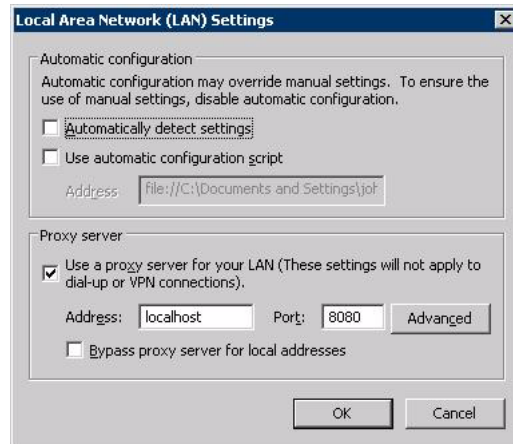
This figure shows the Local Area Network (LAN) Settings dialog.

****

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
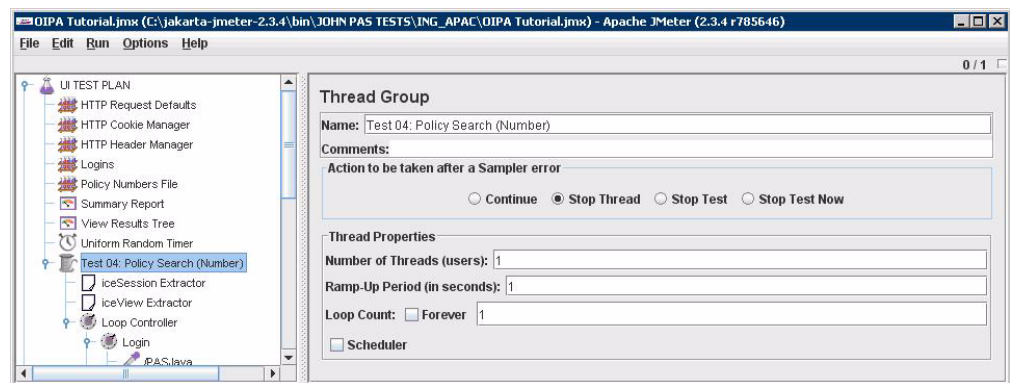
This figure shows the Thread Group section.

****

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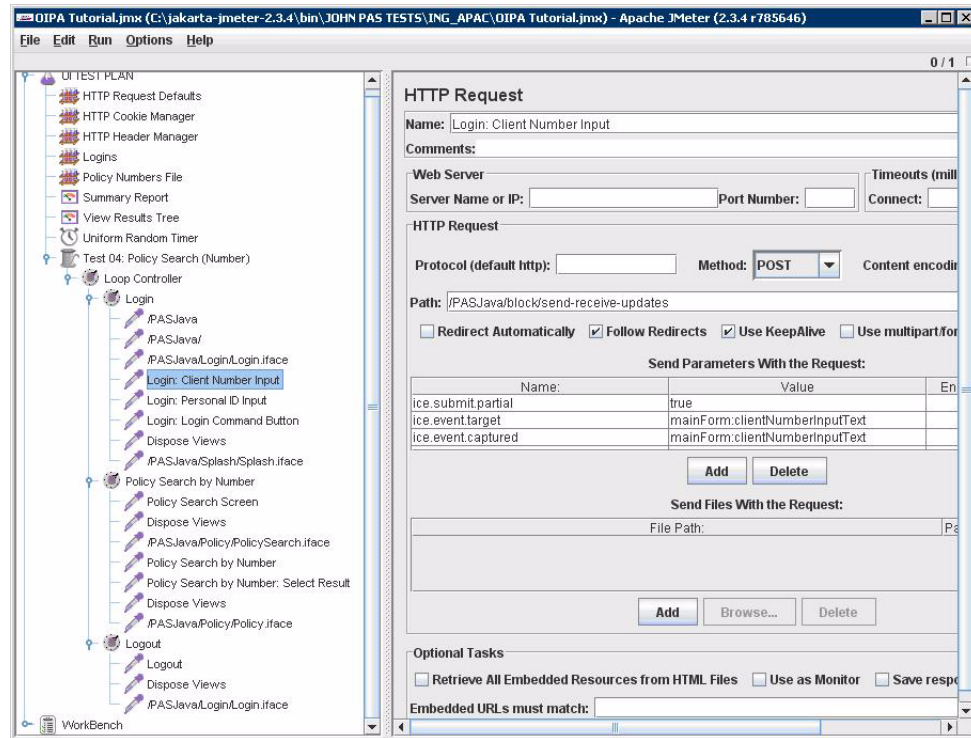
This figure shows the result of the test case, after making the naming changes and grouping the pages for clarity.

****

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
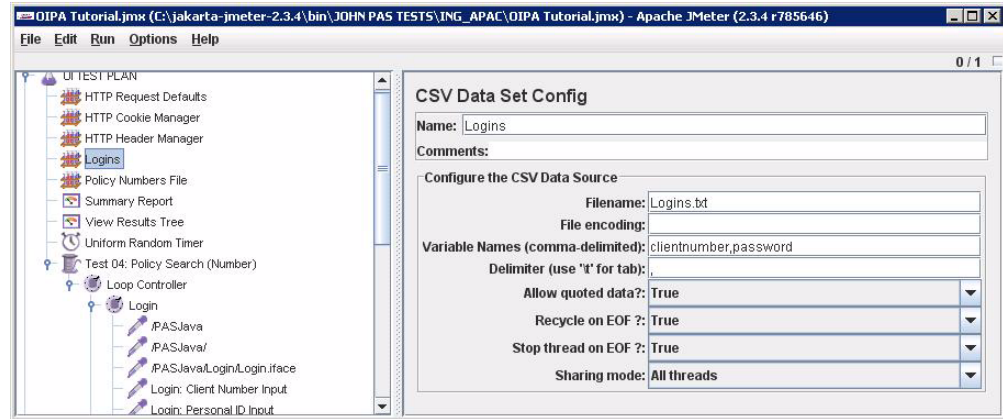
This figure shows the CSV Data Set Config section.

****

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
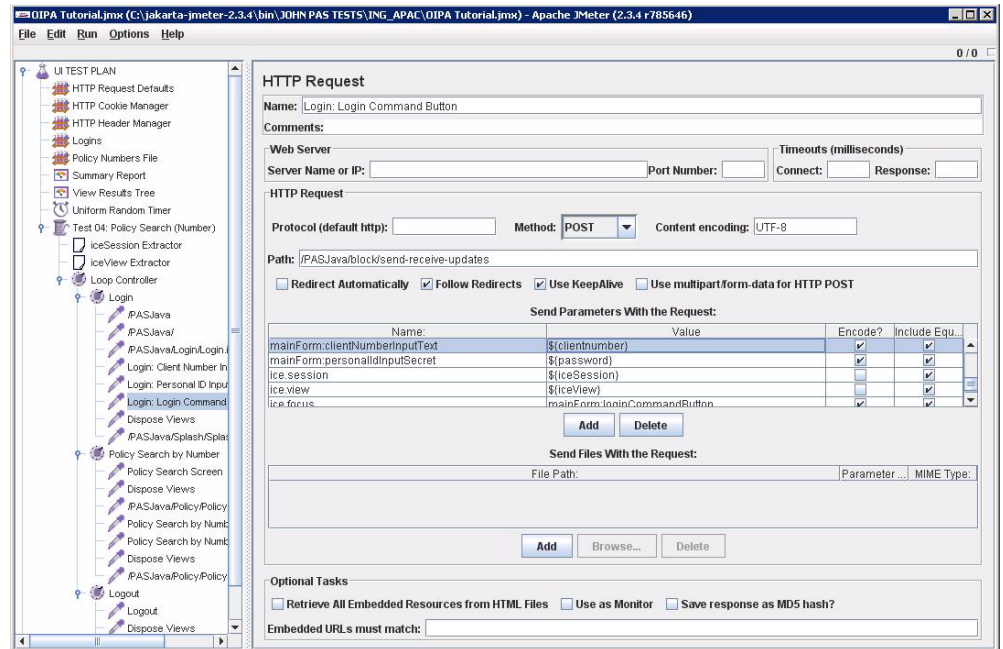
This figure shows the HTTP Request section with the username and password details populated.

****

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
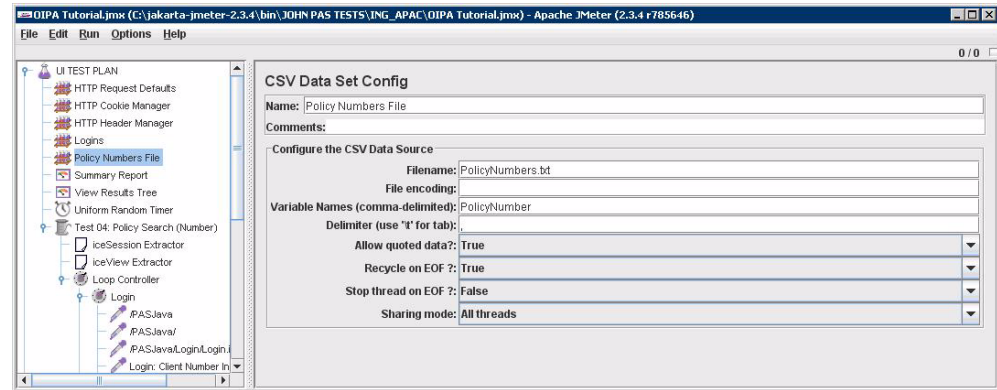
This figure shows the CSV Data Set Config element corresponding to the policy numbers.

****

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
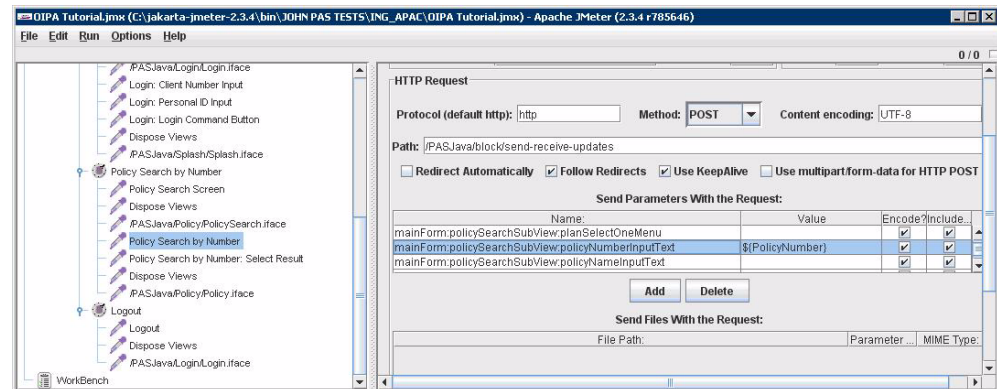
This figure shows the HTTP Request section where you reset the variable for search criteria in the pages where the policy search is executed.

****

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
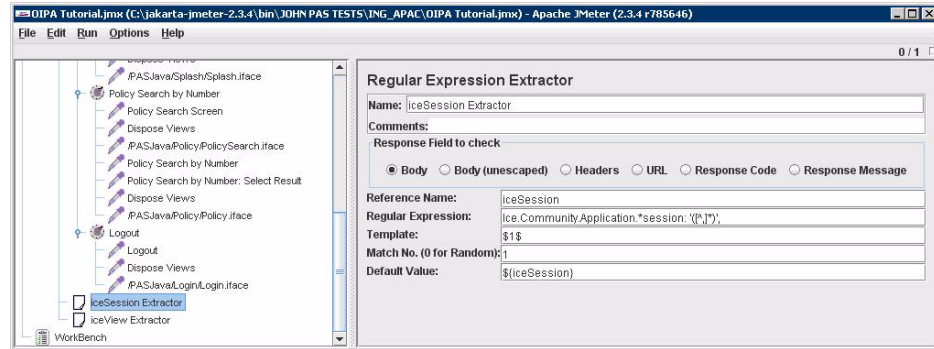
This figure shows the Regular Expression Extractor section where the ice.session value is configured.

****

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
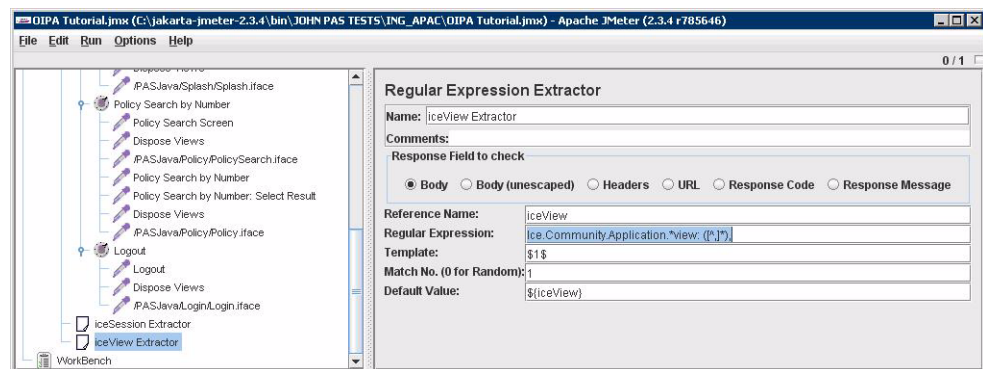
This figure shows the Regular Expression Extractor section where the ice.view value is configured.

****

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
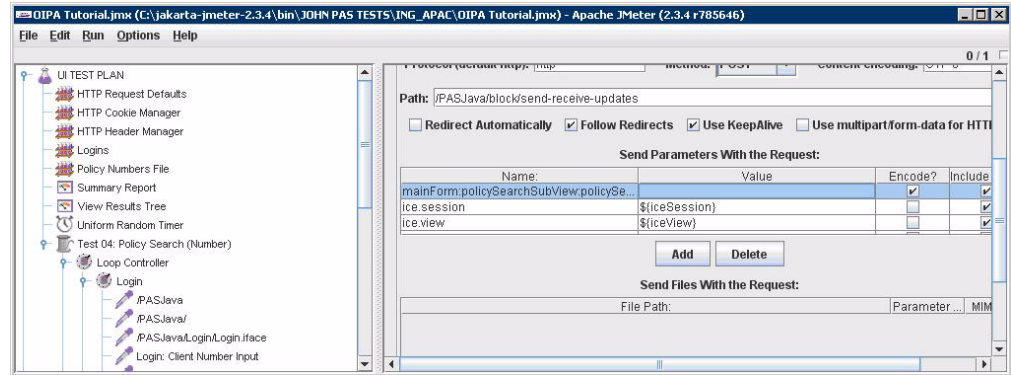
In this figure, the values for ice.session and ice.view are replaced with ${iceSession} and ${iceView}.

****

8. 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
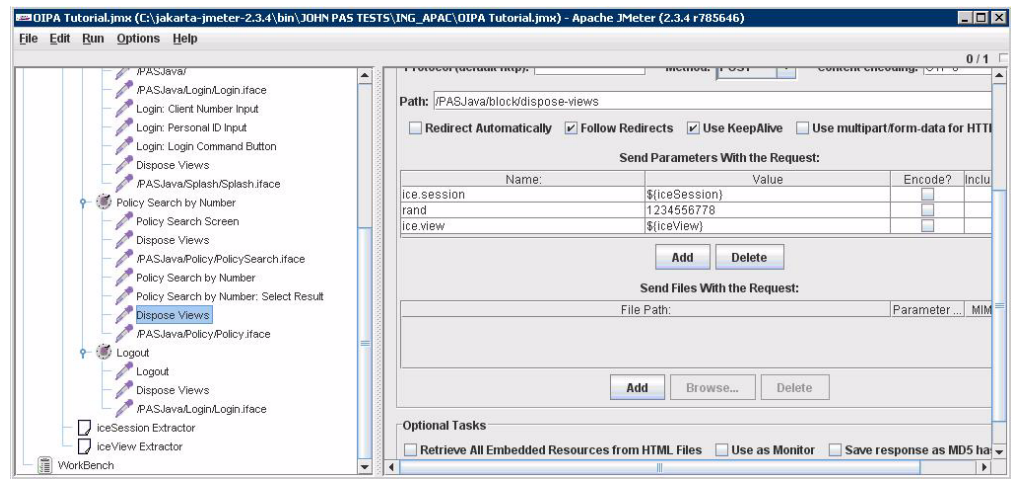
This figure shows the dispose_views page.

****

The input data set file method and the regular expression extractor method parameterization can be expanded to include any value.

Adding Listeners

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

1. Right-click **TEST PLAN**.

A menu is displayed.

2. On the menu, click **Add**, **Listeners**, and then **Summary Report**.

3. 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
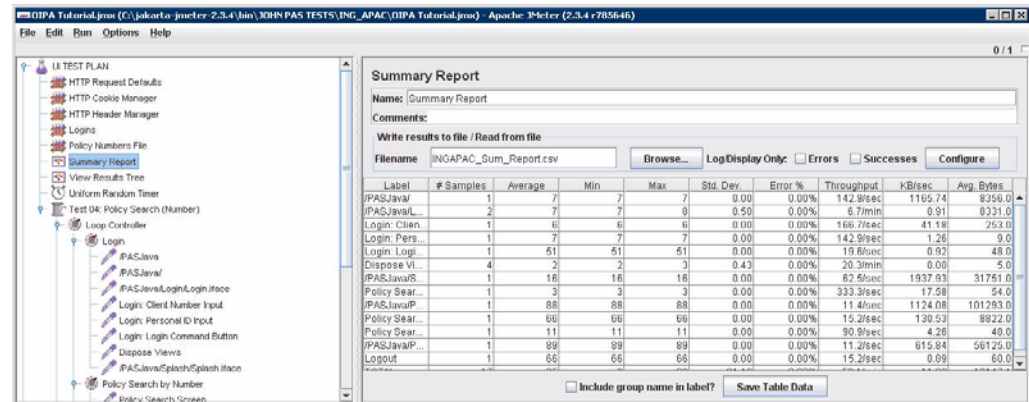
This figure shows the Summary Report page.

****

4. Right-click **TEST PLAN**.

   A menu is displayed.

5. 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
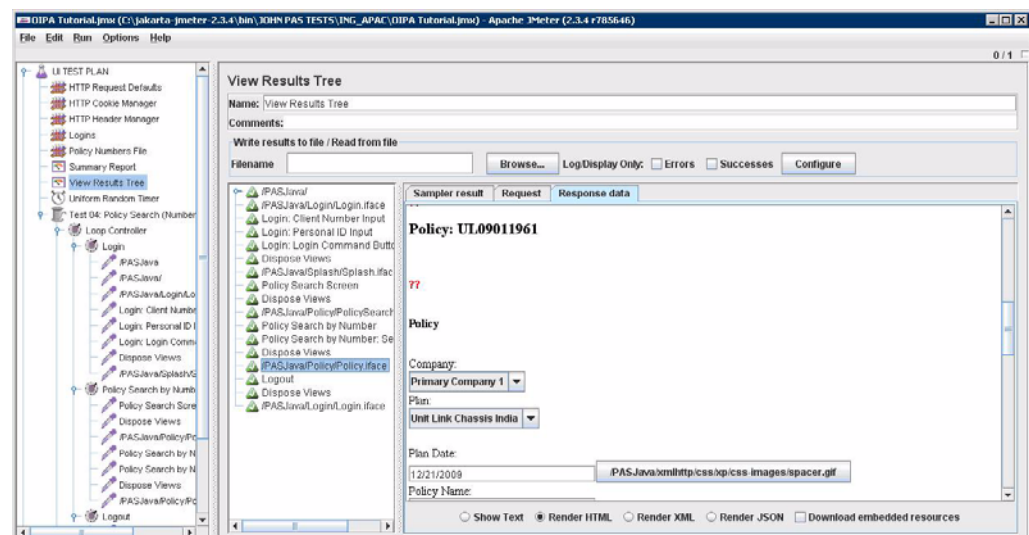
This figure shows the View Results Tree page.

****

Adding Timers

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

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

    The Uniform Random Timer page is displayed.

2. 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
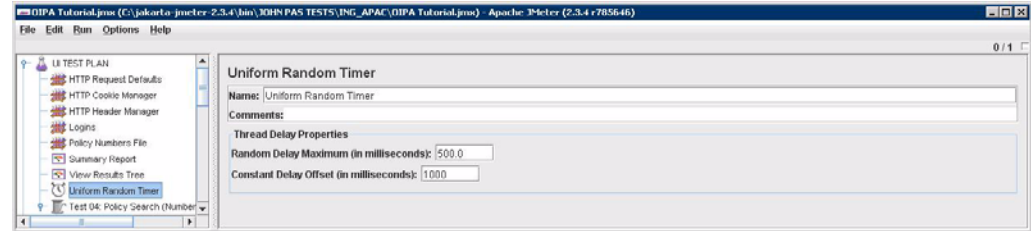
This figure shows the Uniform Random Timer page.

****

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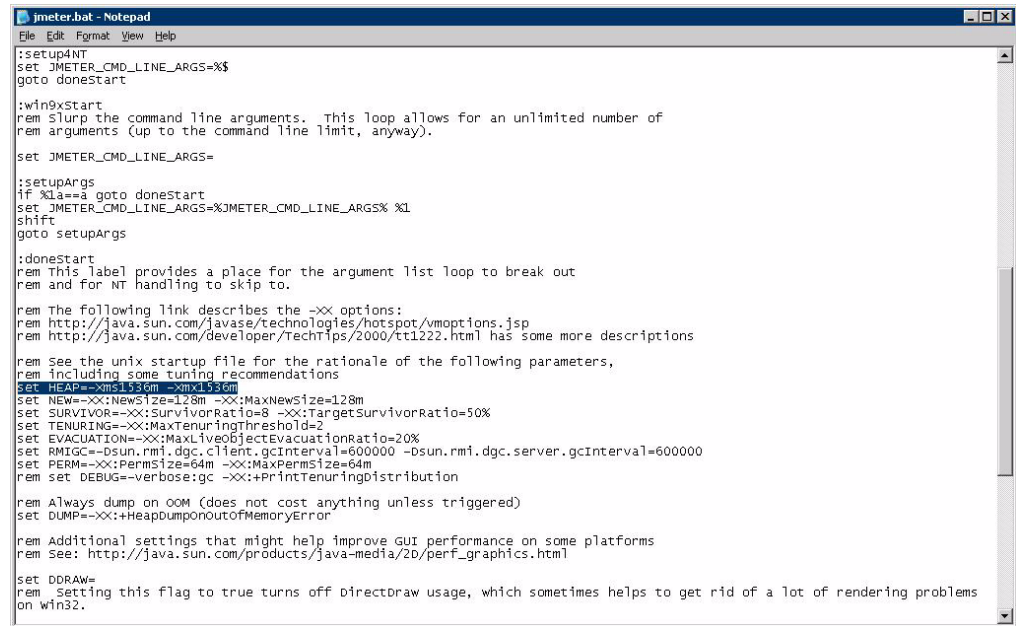
Sometimes if there are many users using Apache JMeter simultaneously, then Apache JMeter may crash. If the system is high end, then you can increase the number of users by altering the heap size.

To change the Java heap size allocation for Apache JMeter:

1. 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.

Figure 20   The JMeter.bat File

This figure shows 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
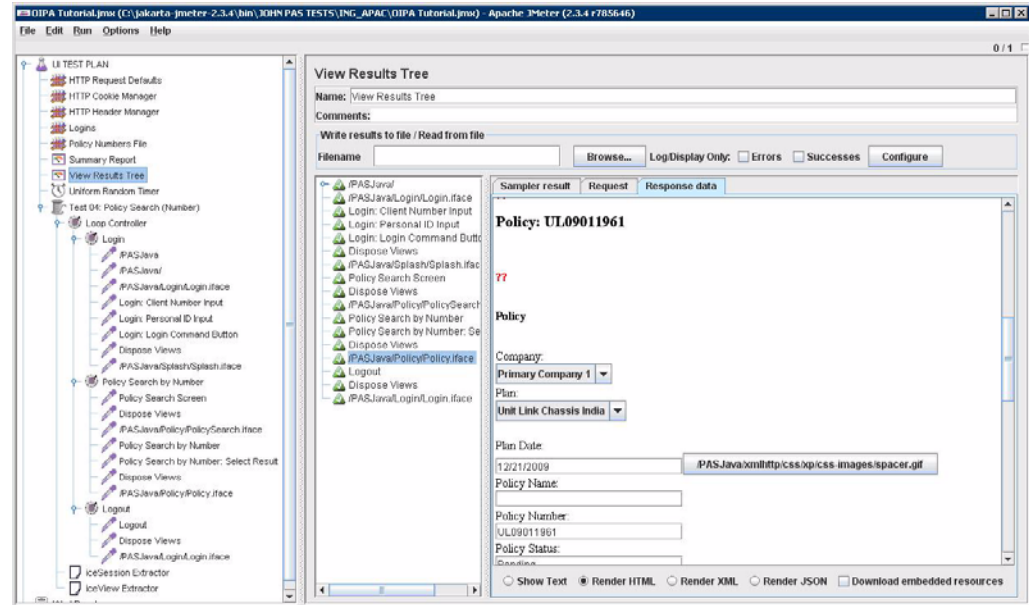
This figure shows the final result of performance testing the OIPA by using JMeter.

****

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
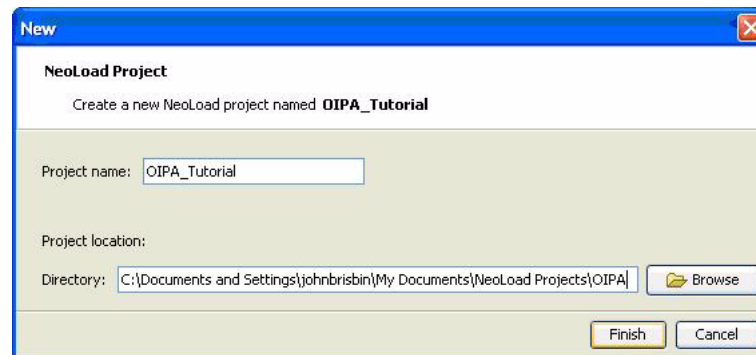
This figure shows the New dialog that is displayed when you create a new project.

3. Click **Finish**.

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

Figure 23   The NeoLoad Window
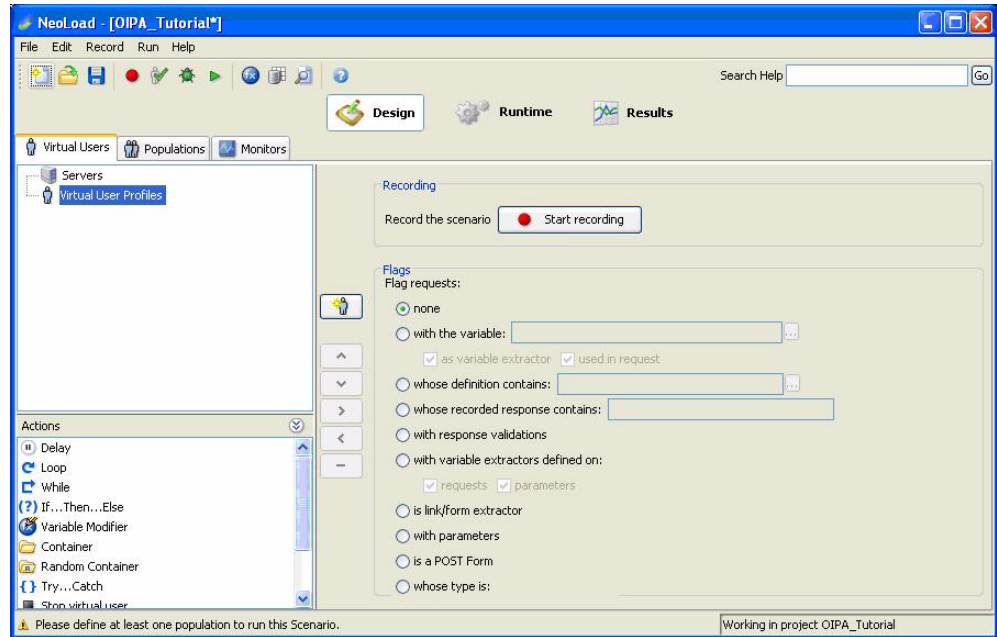
4. Click **Start recording**.

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

This figure shows 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
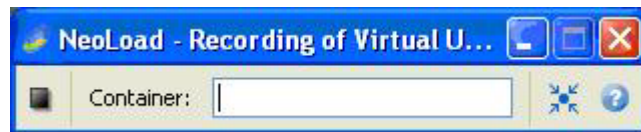
This figure shows the NeoLoad - Recording of Virtual User dialog.

****

    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
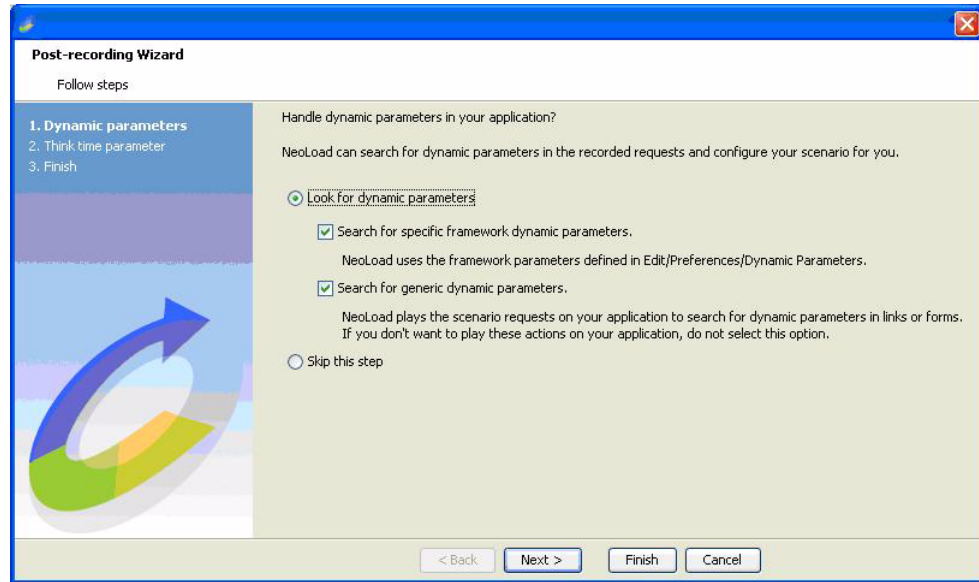
This figure shows the Post-recording Wizard.

****

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
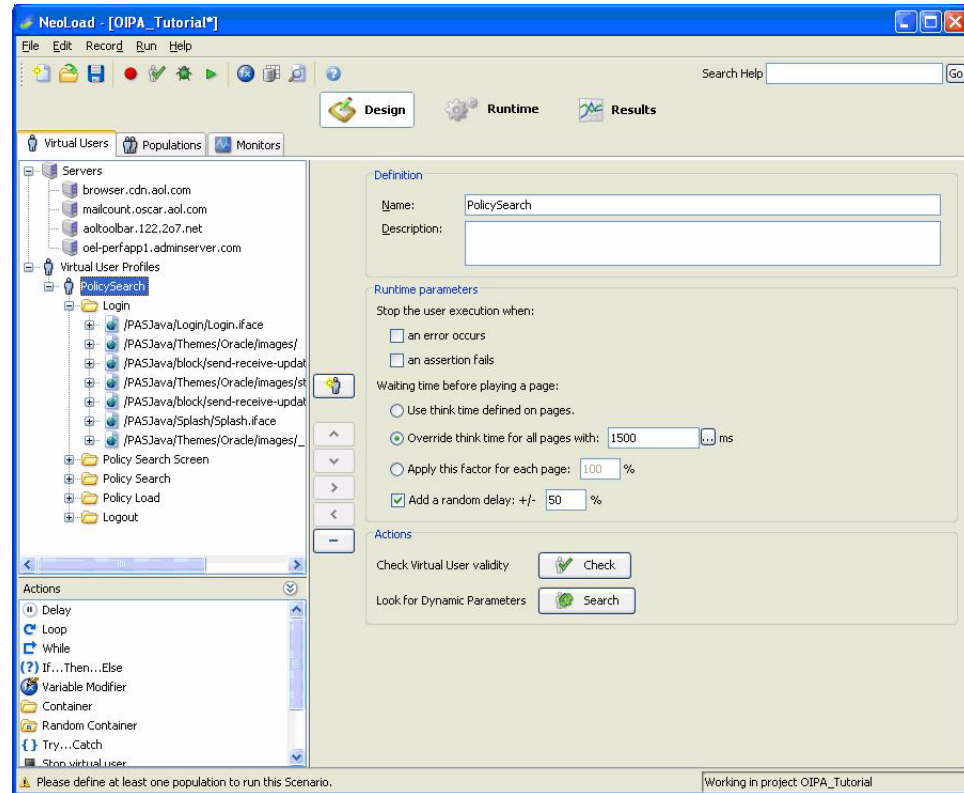
This figure shows the NeoLoad Design window.

****

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
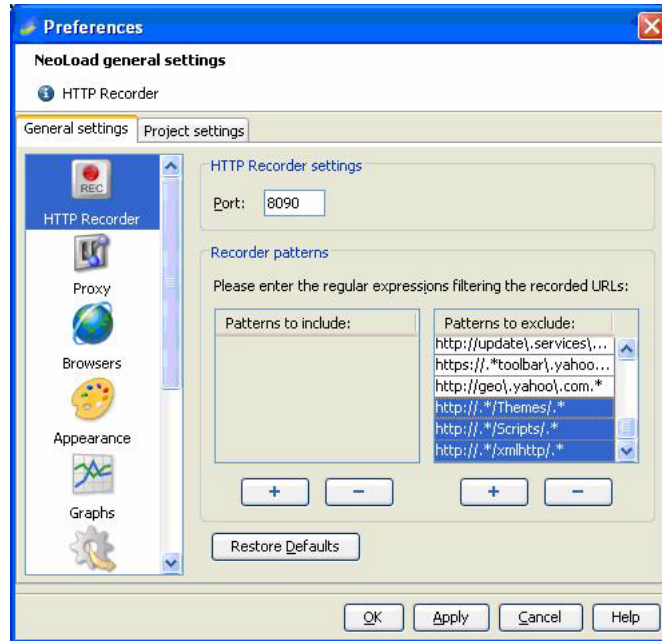
This figure shows the Preferences dialog.

****

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 Check Virtual User Window
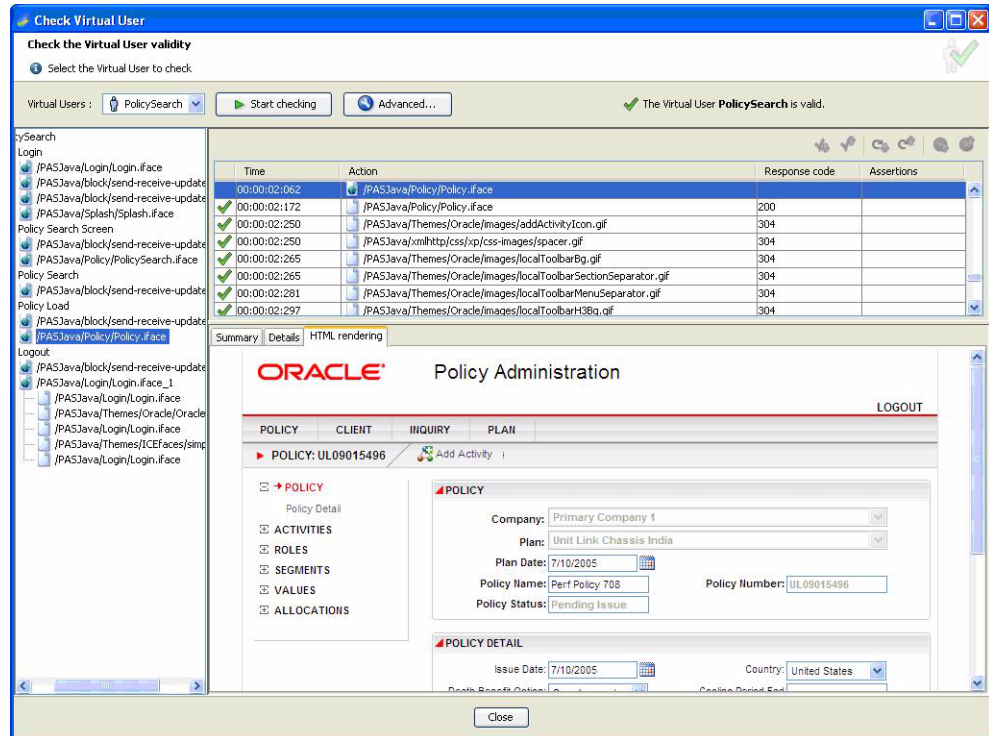
This figure shows the Check Virtual User window.

****

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
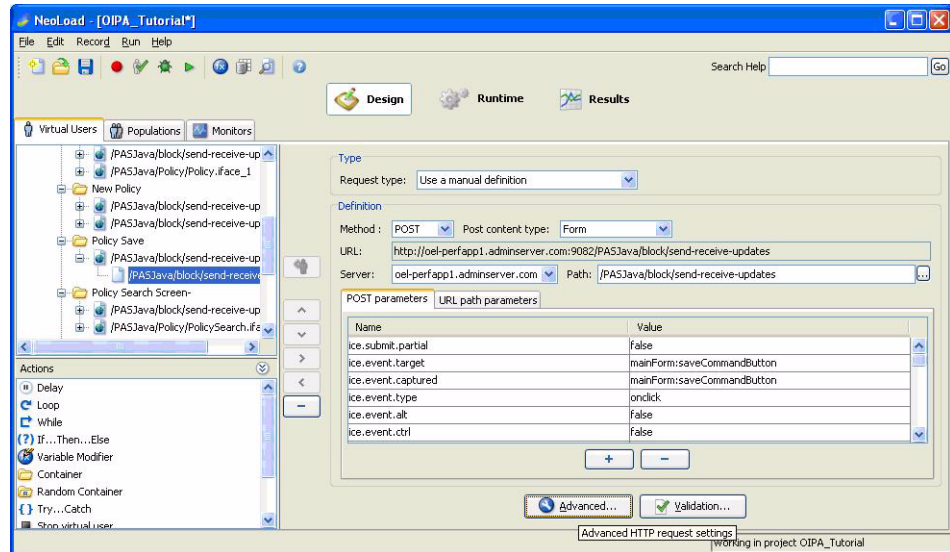
This figure shows the NeoLoad window.

****

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

Figure 31　The Advanced Dialog
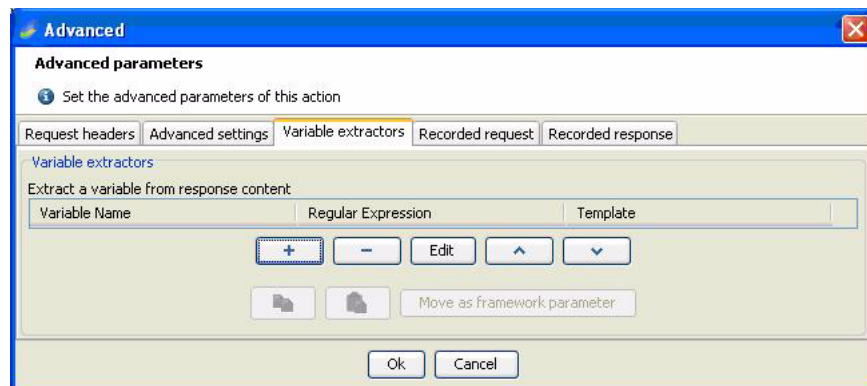
This figure shows the Advanced dialog.

****

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

This figure shows 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
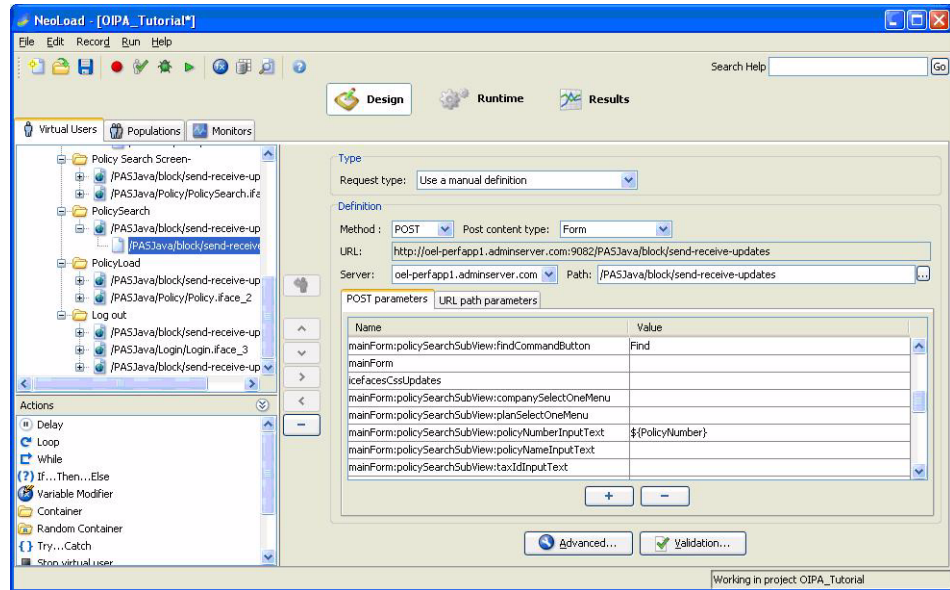
This figure shows how to change the policy number in the Policy Search page.

****

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
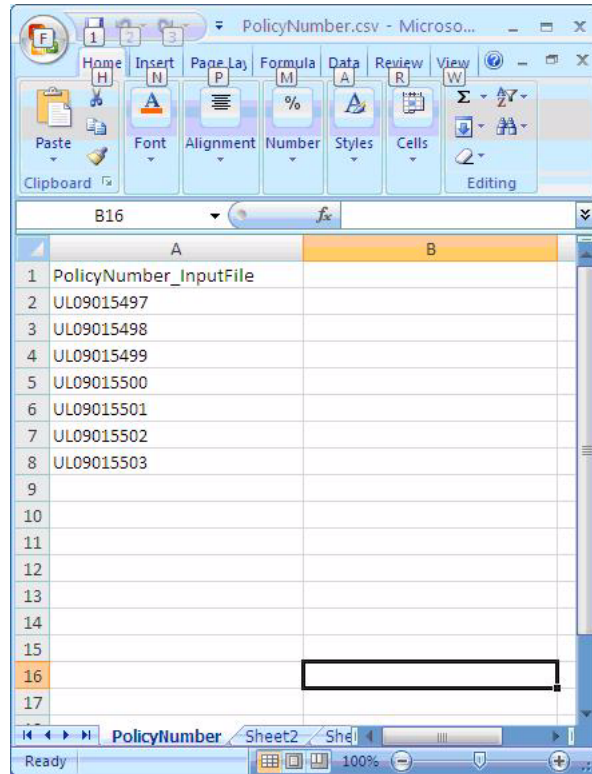
This figure shows the PolicyNumber.csv file.

****

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
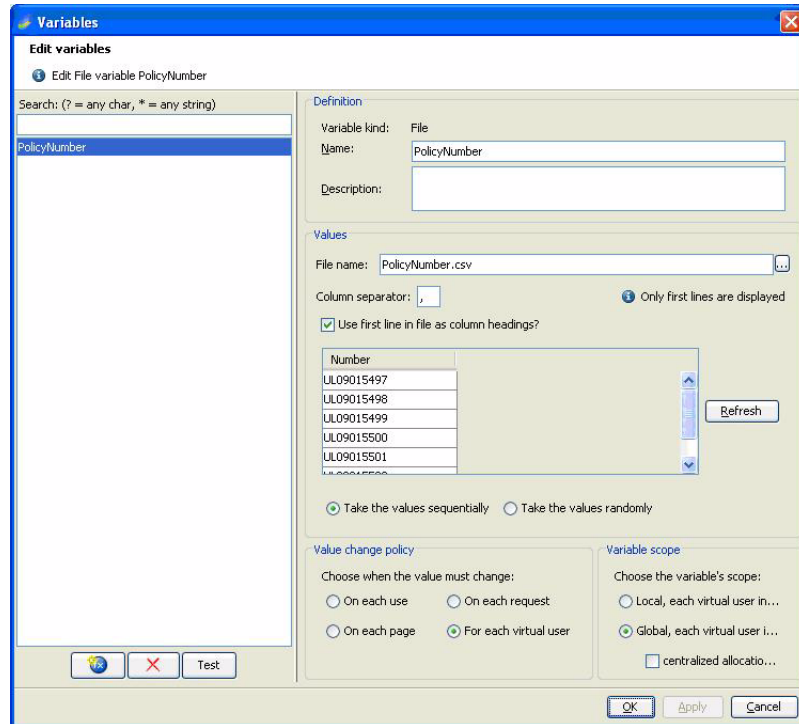
This figure shows the Variables dialog.

****

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
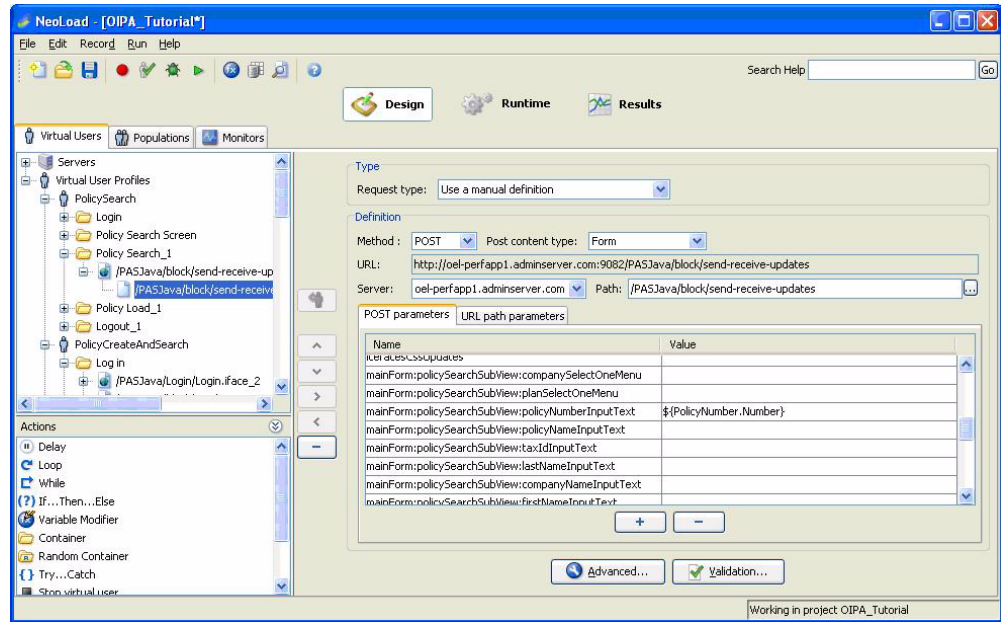
This figure shows the Policy Search screen.

****

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
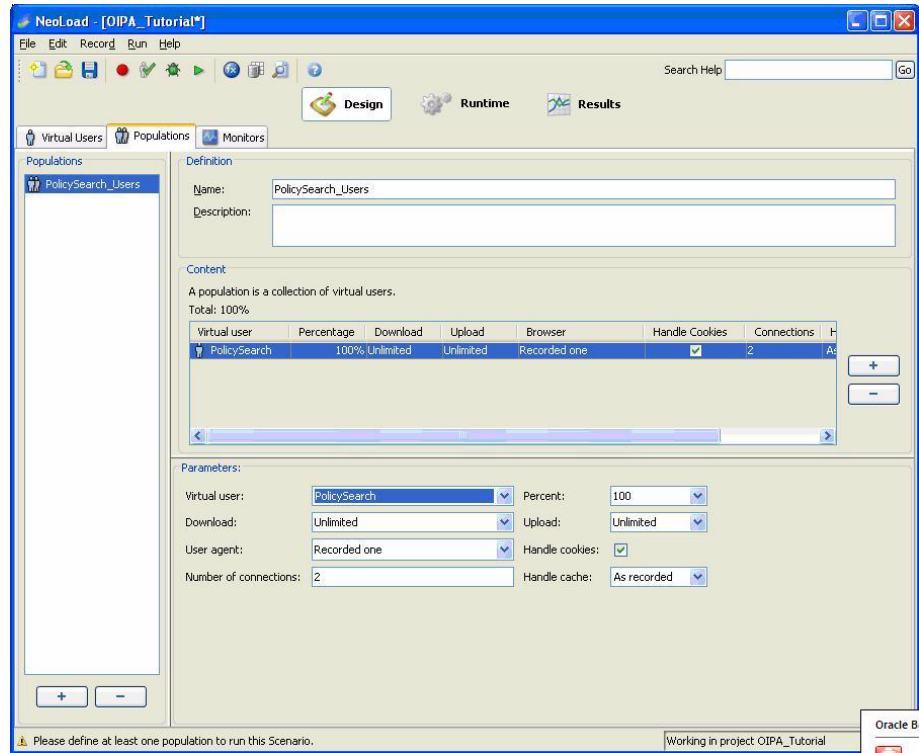
This figure shows the NeoLoad window where you create a population.

****

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
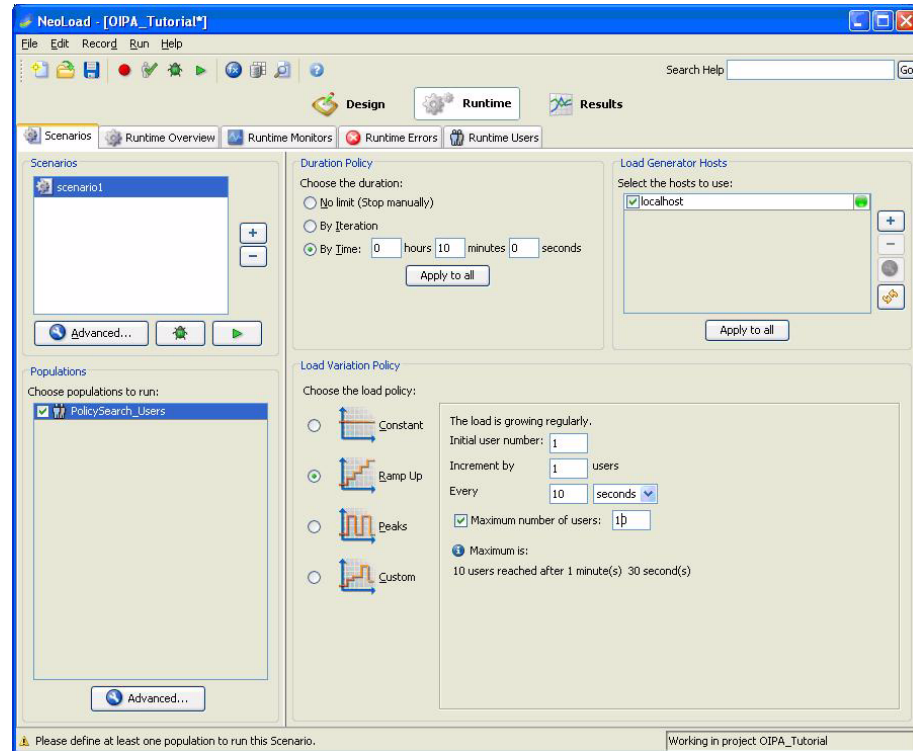
This figure shows the Runtime screen of the NeoLoad window.

****

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   Runtime Statistics of the Test
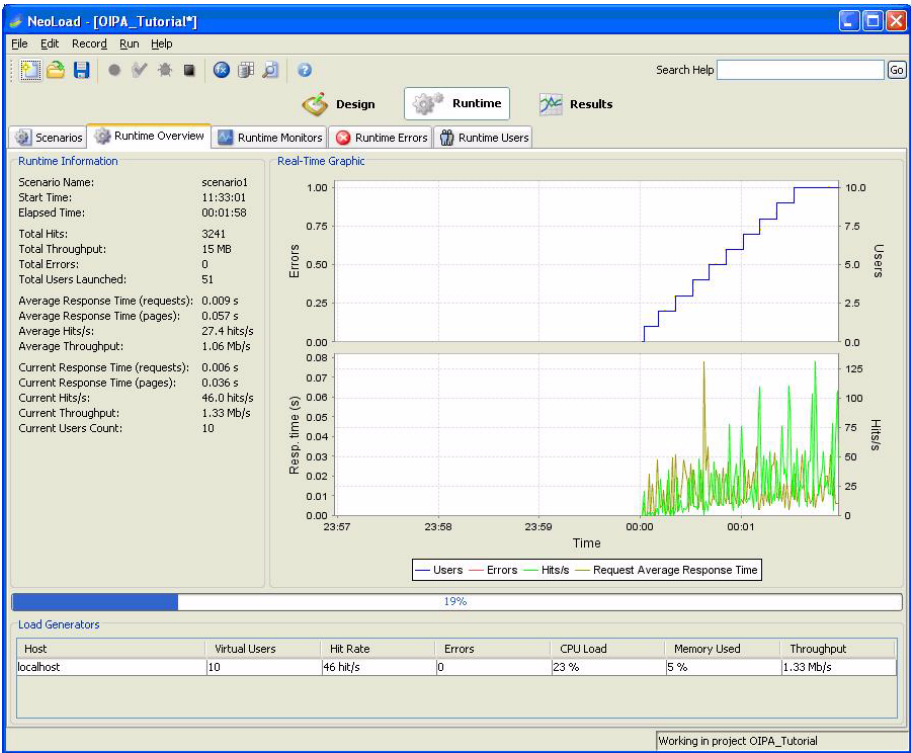
This figure shows the runtime statistics of the test.

****

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

Figure 40   Runtime Monitors
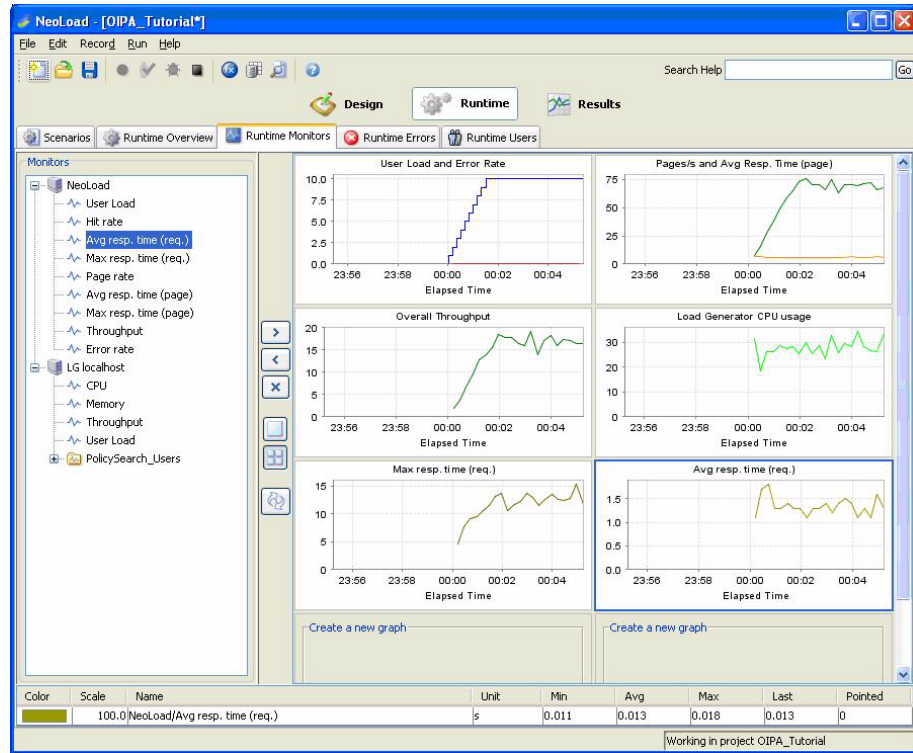
This figure shows the runtime monitors of the test.

****

■   Runtime Errors: displays errors (when applicable)

■   Runtime Users: follows script execution per virtual user, as shown in Figure 41.

Figure 41   Runtime Users
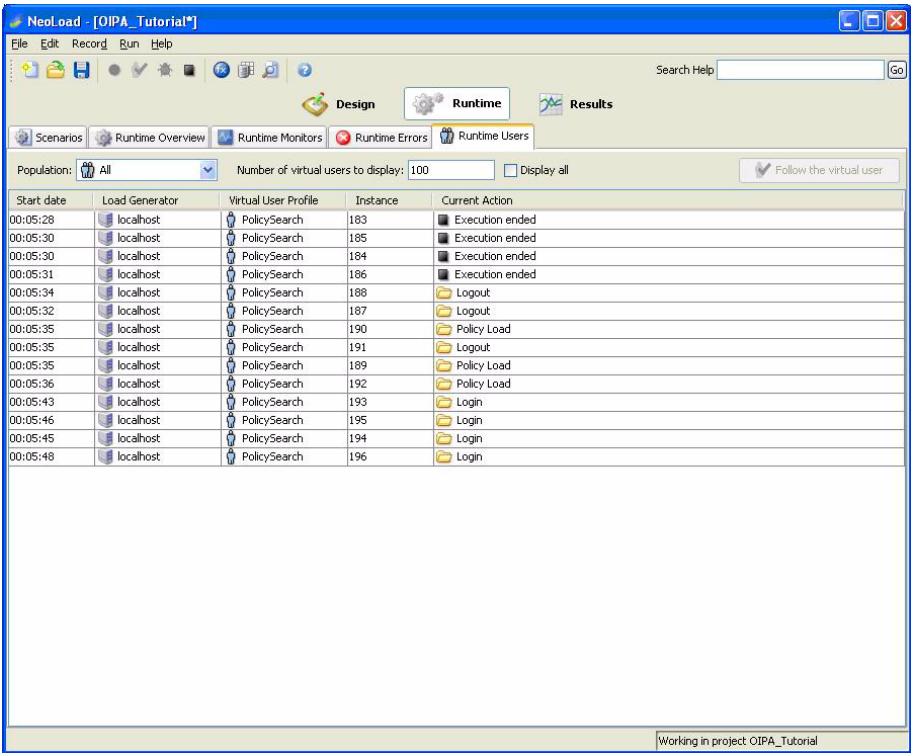
This figure shows the runtime users.

****

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
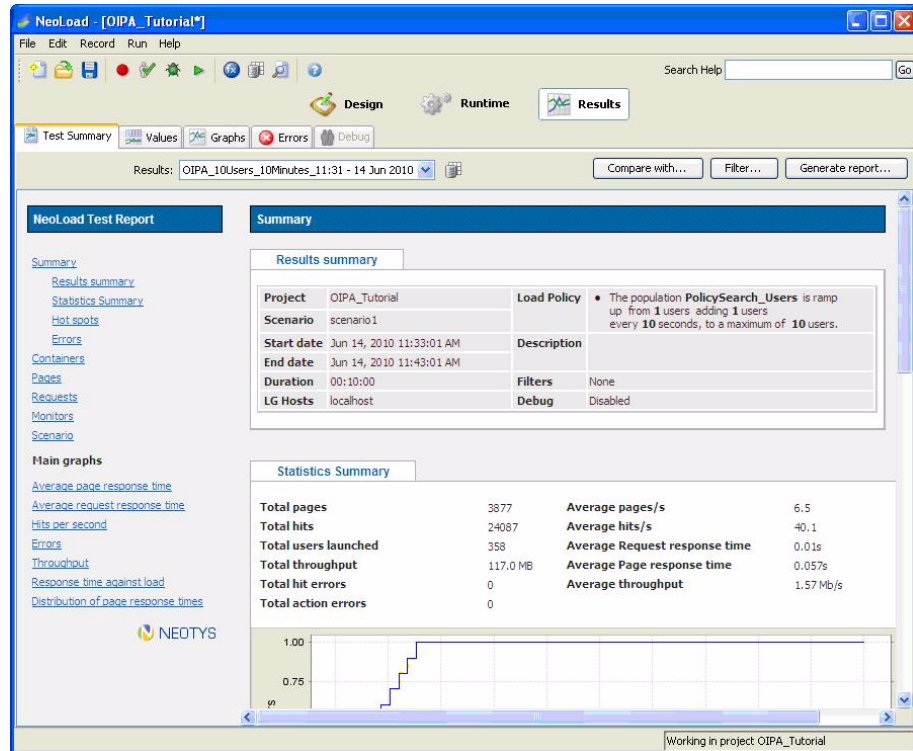
This figure shows the result summary of the test.

****

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
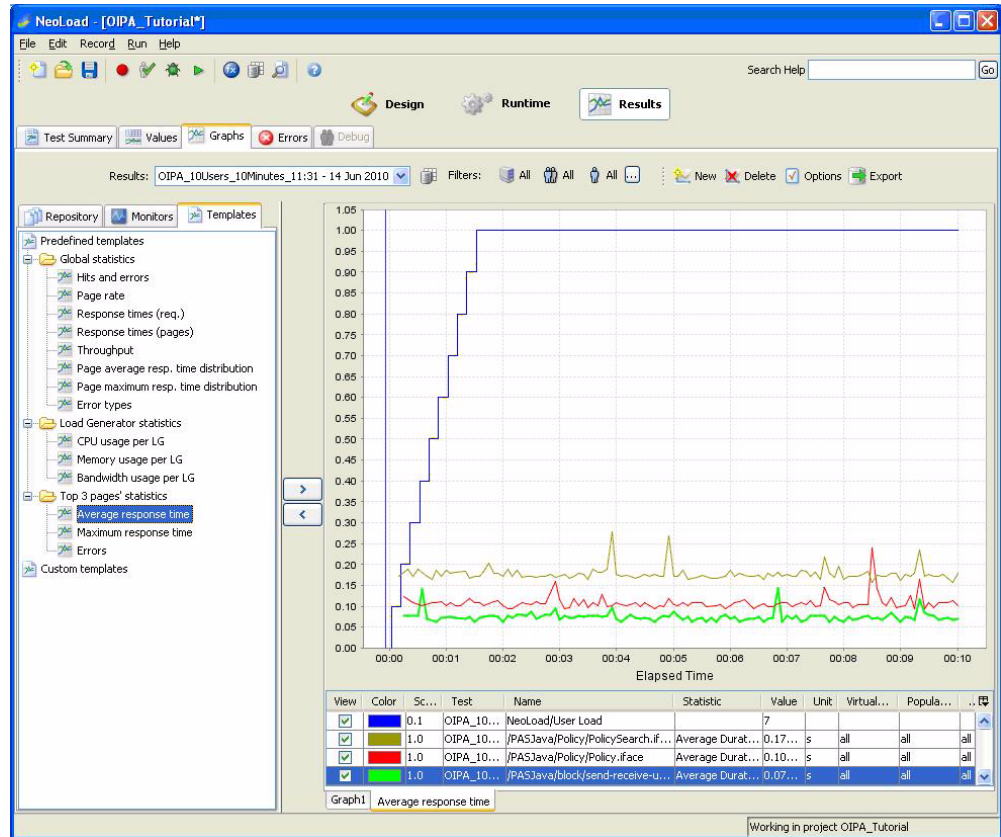
This figure shows the result of the test graphically.

****

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
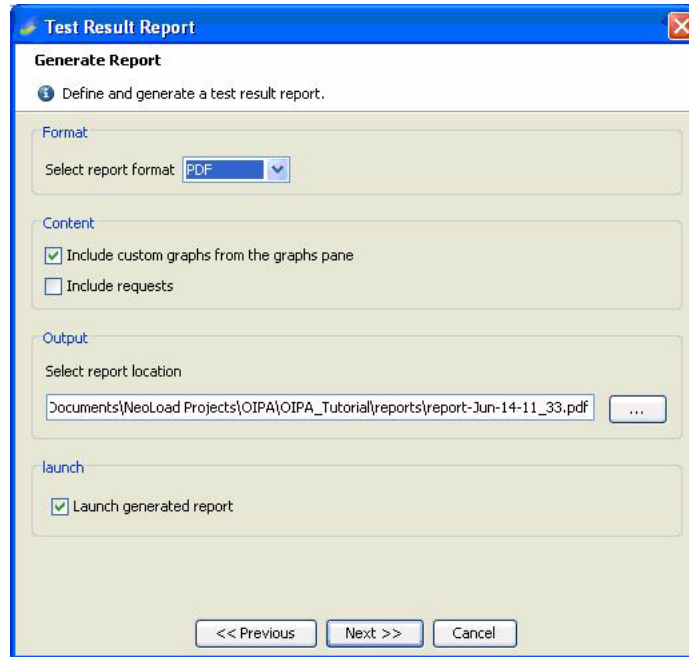
This figure shows the Test Result Report dialog.

****

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
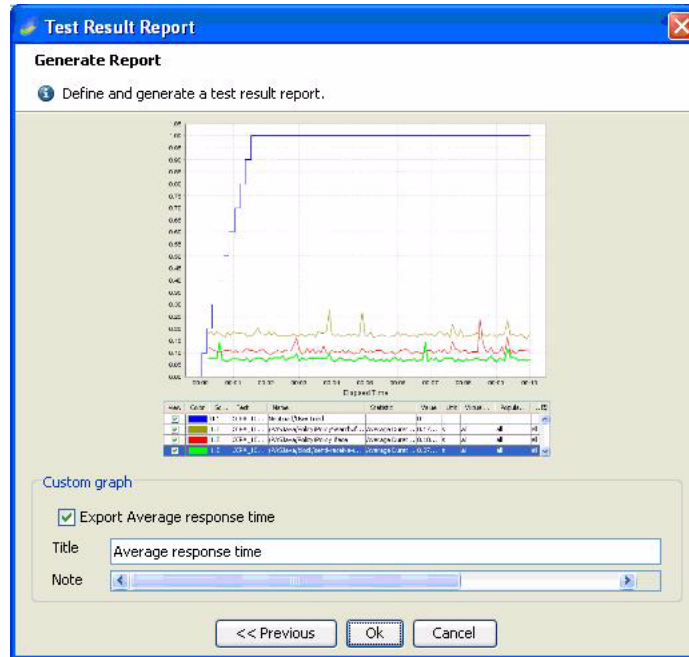
This figure shows the Test Report dialog.

****

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
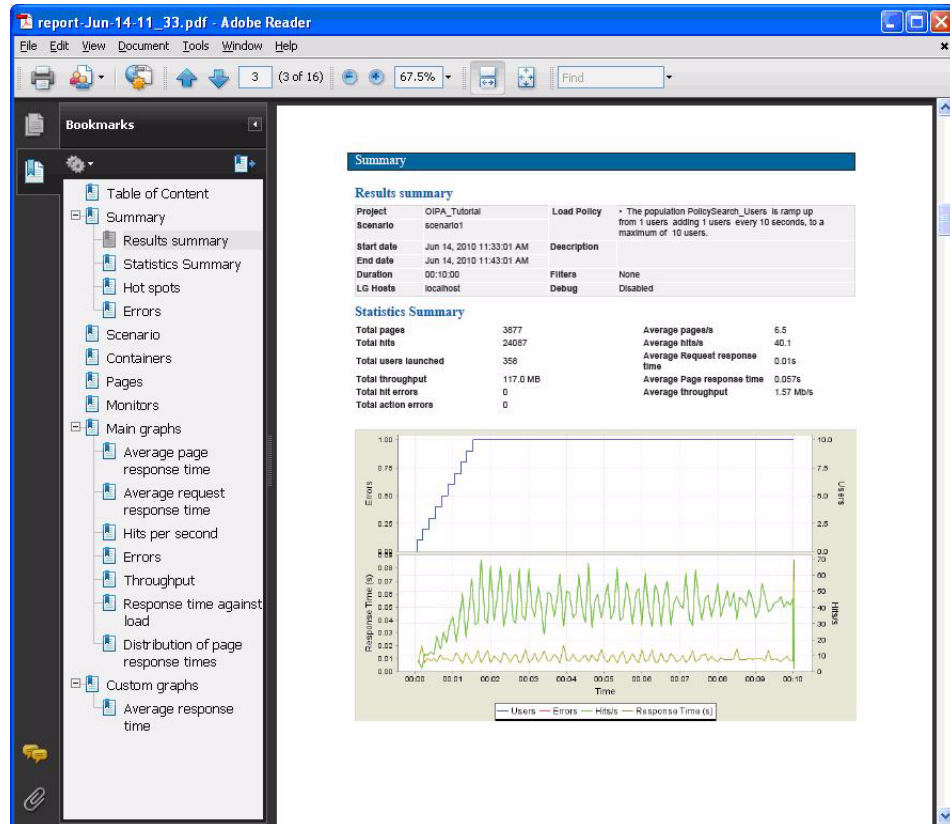
This figure shows the result summary.

****

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
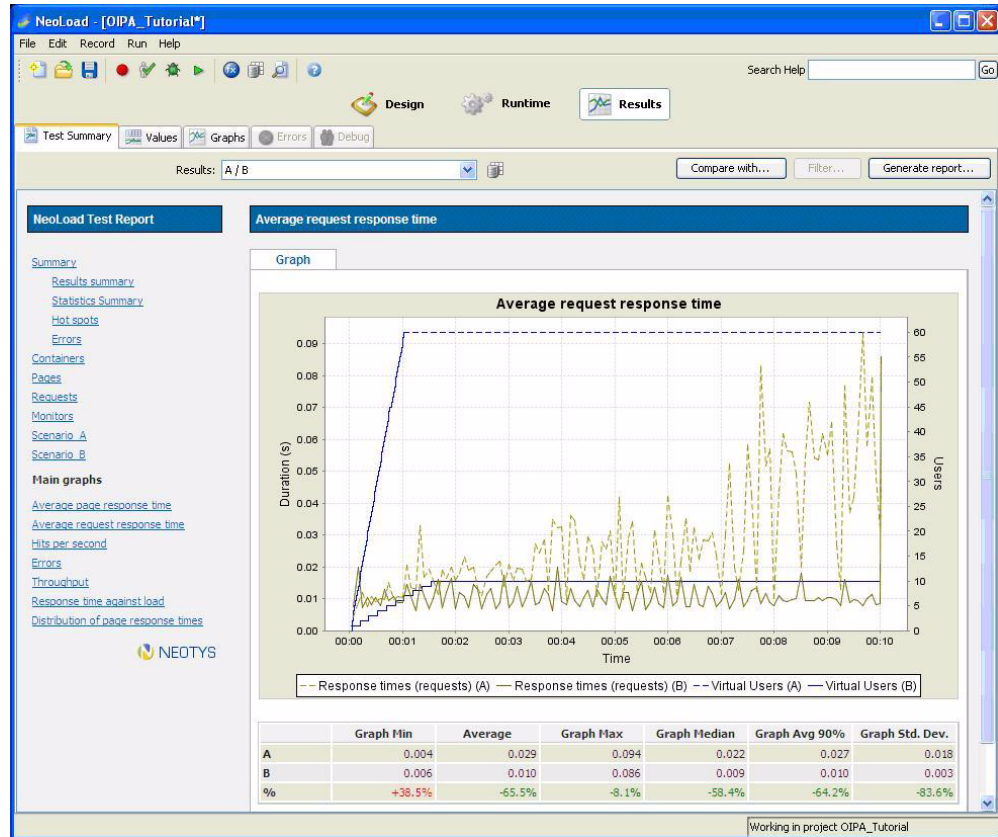
This figure shows comparison result.

`*******************************************************************************************`

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

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.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.