



Pro Apache JMeter

Web Application Performance Testing

Sai Matam
Jagdeep Jain



Pro Apache JMeter

Web Application Performance Testing



**Sai Matam
Jagdeep Jain**

Apress®

Pro Apache JMeter: Web Application Performance Testing

Sai Matam
Pleasanton, California, USA

ISBN-13 (pbk): 978-1-4842-2960-6
DOI 10.1007/978-1-4842-2961-3

Jagdeep Jain
Dewas, Madhya Pradesh, India

ISBN-13 (electronic): 978-1-4842-2961-3

Library of Congress Control Number: 2017951240

Copyright © 2017 by Sai Matam and Jagdeep Jain

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr

Editorial Director: Todd Green

Acquisitions Editor: Celeste Suresh John

Development Editor: Laura Berendson

Technical Reviewer: Nitesh Kumar Jain

Coordinating Editor: Sanchita Mandal

Copy Editor: Kezia Endsley

Distributed to the book trade worldwide by Springer Science+Business Media New York,
233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail
orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC
and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc).
SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-2960-6. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

I dedicate this book to my wife, Jyothi, for inspiration, support, and for single-handedly running around kids and various chores while I tapped at the keyboard.

—Sai Matam

I dedicate this book to my parents, who always motivated me to do things differently, and to my sisters and my wife; without their support, I would not be able to manage a tight schedule on and off work.

—Jagdeep Jain

Contents at a Glance

About the Authors.....	xvii
About the Technical Reviewer	xix
Acknowledgments	xxi
Introduction	xxiii
■ Chapter 1: Foundation	1
■ Chapter 2: Performance Testing Primer	3
■ Chapter 3: Your First JMeter Test.....	13
■ Chapter 4: JMeter Test Script Recorder	25
■ Chapter 5: JMeter Test Plan Components	35
■ Chapter 6: Distributed Testing.....	167
■ Chapter 7: JMeter Best Practices.....	179
■ Chapter 8: Troubleshooting JMeter	197
■ Chapter 9: JMeter Plugins.....	211
■ Chapter 10: JMeter Recipes	221
■ Chapter 11: Case Study: Digital Toys Inc.....	243
■ Chapter 12: Performance Dashboard	303
■ Chapter 13: Appendix A: Setting Up JMeter	315
■ Chapter 14: Appendix B: Setting Up Digital Toys Inc.....	327
Index.....	333

Contents

About the Authors.....	.xvii
About the Technical Reviewer	xix
Acknowledgments.....	xxi
Introduction	xxiii
■ Chapter 1: Foundation	1
Why Performance Testing?.....	1
Why JMeter?	2
■ Chapter 2: Performance Testing Primer	3
Performance Testing	3
Response Time	3
Throughput.....	4
Utilization.....	5
Robustness.....	5
Scalability	5
User Perception	5
Cost.....	5
Types of Performance Tests	5
Stress Tests	5
Load Tests.....	6
Peak Load Tests.....	6
Soak Tests or Endurance Tests	6
Scalability Tests.....	6
Capacity Tests.....	6

■ CONTENTS

Spike Tests and Burst Capacity	7
Performance Smoke Tests.....	7
High Availability Test/Fail-Over Tests	7
The Performance Test Environment	9
The Need for Separate Performance Environment.....	9
The Performance Environment Should Be Like the Production Environment.....	9
The Performance Environment Should Be Isolated	9
Performance Testing Tools.....	10
The Performance Testing Strategy Document.....	10
Performance Requirements.....	10
Performance Goals	10
Performance Test Suite	10
Performance Reporting and Analysis	11
Performance Tuning.....	11
Conclusion.....	12
■ Chapter 3: Your First JMeter Test.....	13
Components of a JMeter Test.....	13
Test Plan	13
Thread Group.....	13
Controller.....	14
Sampler	14
Listener.....	14
Timer	14
Assertions.....	14
Config Element	14
Pre-Processors	15
Post-Processors	15
Order of Component Execution.....	15
Simple JMeter Test.....	17
GUI Mode	21

Non-GUI Mode	22
Executing a Single Test.....	23
Proxy Server Setting.....	24
Start JMeter in Server Mode	24
Stop/Shutdown JMeter.....	24
Conclusion.....	24
■ Chapter 4: JMeter Test Script Recorder	25
JMeter WorkBench	25
JMeter Recording Controller	28
Browser Proxy Settings	28
Recording Example	29
Conclusion.....	34
■ Chapter 5: JMeter Test Plan Components	35
Test Plan.....	35
Configuration	36
Serial Execution of Thread Groups	37
Parallel Execution of Thread Groups.....	39
User Defined Variables.....	41
Thread Group.....	42
Thread Properties	43
Scheduler	49
Action After Sampler Error.....	53
Pre-Processors.....	61
HTTP URL Re-Writing Modifier.....	61
Controller.....	63
Simple Controller.....	64
Transaction Controller.....	65
Loop Controller	68
Runtime Controller.....	70

■ CONTENTS

Throughput Controller.....	72
Once Only Controller.....	74
Interleave Controller	76
Random Controller.....	81
Random Order Controller.....	81
Switch Controller.....	82
ForEach Controller.....	85
If Controller.....	87
Timers	92
Constant Timer	93
Gaussian Random Timer.....	97
Uniform Random Timer.....	99
Constant Throughput Timer	101
Synchronizing Timer	103
Sampler	105
HTTP Request	106
Assertions	128
Response Assertion	128
Listener	138
View Results Tree	141
View Results In Table.....	147
Aggregate Report.....	150
Post-Processors	152
Regular Expression Extractor	152
Properties and Variables	157
Comparison of Properties and Variables.....	157
User Defined Variables.....	161
Using the Command Line to Initialize Properties.....	164
Conclusion.....	165

■ Chapter 6: Distributed Testing.....	167
Distributed Testing Using JMeter	167
Prerequisites	167
Configuration.....	168
Running the Test.....	170
GUI Mode	171
Non-GUI Mode	172
RMI Port.....	172
Sample Sender Mode	173
Unreachable Remote Hosts	176
Limitations.....	177
Conclusion.....	177
■ Chapter 7: JMeter Best Practices.....	179
HTTP Request Defaults.....	179
Follow Redirects.....	180
Cookie Manager	182
Cache Manager	185
JMeter Using Maven	185
Passing Variables Across Thread Groups	187
Running Parallel Thread Groups	191
Using External File for Parameterizing User Login.....	192
Customizing Properties	194
Monitor JMeter Resource Usage	194
Standard Test Plan Templates	195
Conclusion.....	196

■ Chapter 8: Troubleshooting JMeter	197
Ensure Permissions.....	197
Log File.....	197
Log Level	198
HTTP Protocol Logs	199
GUI Logs	199
Clear GUI Logs	200
Remote Host Exception	200
Connect Exception.....	201
Solving Proxy Servers Problems	202
HTTP Basic Authentication	204
Using HTTP Header Manager	205
Using the HTTP Authorization Manager.....	206
Debug Test Faster.....	207
Out of Memory Error.....	209
Conclusion.....	210
■ Chapter 9: JMeter Plugins.....	211
PerfMon.....	211
Download the Plugin.....	212
Start the PerfMon Agent	213
Non-GUI Mode	218
Conclusion.....	219
■ Chapter 10: JMeter Recipes	221
JDBC Performance Testing	221
Install MySQL.....	221
Install JDBC Driver.....	224
JDBC Test Plan.....	224
FTP Performance Testing	228
REST/JSON Performance Testing	230

AJAX Performance Testing	232
Mobile Performance Testing.....	234
Simulating Mobile Devices	234
Simulating Network Speed.....	234
JMeter to Record User Actions	235
SOAP Performance Testing.....	237
Install SOAPUI	238
Conclusion.....	242
■ Chapter 11: Case Study: Digital Toys Inc.	243
The Need for Speed.....	243
Addressing the Problem	244
Performance Goals	244
Performance Test Specification.....	244
Tool Selection	247
Test Environment.....	247
Test Data Preparation	248
User Load Pattern.....	248
Application Build.....	249
Using JMeter	249
Test Script Development.....	250
Validation of Test Steps	259
Passing Variables Between Samplers	263
Running Tests with Multiple Users	265
Implementing Actual User Behavior	271
Results Metrics.....	275
Organizing Tests	285
Combining Multiple Tests	293
Questions	297
Using Distributed Environment.....	298
Performance Testing and Tuning Cycle	301

■ CONTENTS

Outcome	301
Conclusion	302
■ Chapter 12: Performance Dashboard	303
APDEX.....	303
Configuration.....	303
JMeter Properties.....	304
APDEX.....	304
Global Graph Properties.....	304
Specific Graph Properties.....	305
Generating Graphs.....	306
Performance Dashboard Graphs	306
Conclusion.....	313
■ Chapter 13: Appendix A: Setting Up JMeter	315
MacOSX	315
Download JDK	315
Install JDK.....	316
Set Up the Environment Variable	316
Download JMeter	317
Set Up JMeter.....	317
Windows	318
Download JDK	318
Install JDK.....	319
Set Up the Environment Variable	320
Download JMeter	321
Set Up JMeter.....	321
Linux	322
Install JDK.....	322
Set Up the Environment Variable	323
Download JMeter	323
Set Up JMeter.....	324

■ Chapter 14: Appendix B: Setting Up Digital Toys Inc.	327
Running Digital Toys Web Application	327
Start the Web Application	327
Start with URL Rewriting Enabled	330
Clean Up	331
Index.....	333

About the Authors



Sai Matam has more than 20 years of diverse experience in software development, including significant experience in performance testing and tuning. He has worked on tuning Java and web applications with many millions of page visits.



Jagdeep Jain has more than a decade of experience in software quality assurance and testing. He holds a degree in Computer Science and Engineering. He is a firm believer and advocate of test automation, and he has used Apache JMeter extensively.

About the Technical Reviewer



Nitesh Kumar Jain has over a decade of experience in the software testing world. He has an M.Tech in Information technology from IIITM Gwalior and a B.E. in Computer Science and Engineering. He is a keen technology learner with an “let’s automate everything” attitude. He is also an ISTQB certified test manager, technical test analyst, test analyst, and Agile test engineer and loves to make Java-Swing based tools that can help with software testing. He also spent five years on performance testing and test automation.

Acknowledgments

We want to thank the following people who have helped us make sure that the book is useful by providing timely feedback on our chapters, testing JMeter scripts, and finding bugs in our sample web application. Without them, it would have been tough to create a good quality book.

Ai Yu, Alap Shah, Amit Devgan, Anand Sinha, Anil Ramesh Malleboyina, Anil Wadghule, Beejal Vibhakar, Belal Ansari, Bhushan Gupta, Chakradhar Kommera, Charan Das Thota, David Livingstone Gangarapu, Deepa Mahendraker, Dheeraj Sah, Etender Naini, Ganesh Somaka, Gomtesh Gandhi, Govardhan Aliseri, Haridev Vengateri, Harshad Savot, Harshvardhan Vipat, Hemanth Presingu, Mangesh Lunawat, Manjula Gundugollu, Manjula Kavadi, Manohar Gone, Nikhil Agrawal, Nitish Shirasath, Pankaj Saraf, Piyush Singh, Prashanth Abbagani, Prasoon Kumar, Raj Gopal Marripalli, Rama Gangadhar Mekala, Ram Katru, Roshan Iqbal, Ruth Rajitha Gangarapu, Sanjeev Kumar, Satish Salandri, Satyapal Reddy Panyala, Sharon Annese, Shravan Goli, Shyam Palleti, Shyam Palreddy, Sridhar Throvagunta, Srikanth Ganapavarapu, Srinivas Nagandla, Srinivas Reddy Gaddam, Sudeep Tripathy, Suneeta Donepudi, Sunil Kumar, Sunil Potti, Swamy Das, Tapan Upadhyay, Tarak Joshi, Vidhut Singh, Vijay Pasupuleti, Yogesh Sharma and Yogesh Yadhav.

We are very thankful to the editorial team at Apress and the technical reviewer for having various checkpoints in place and providing us with useful feedback in a timely manner, all of which have made this book more useful for readers.

Introduction

This book is intended to get beginners up and running with performance testing using JMeter. This book provides step-by-step guidance and covers advanced topics for the experienced engineer. Each chapter is clearly marked with the topics it covers, thereby allowing the reader to skip chapters if appropriate.

Chapter 1 is the foundation of the book where we discuss why performance testing is needed and why we should user JMeter.

Chapter 2 is general-purpose chapter that covers the performance testing methodology.

Chapters 3 through 5 cover specific topics in JMeter (test plan, thread group, pre-processors, controller, timers, sampler, assertions, listeners, post-processors, properties, and variables). By going through these chapters, you will gain an understanding on how to user JMeter for your performance needs.

Chapters 6 through 10 deal with distributed testing, a few advance concepts of JMeter, and troubleshooting tips that will be useful in some projects.

Chapter 11 contains the case study of a sample web application called Digital Toys Inc. This chapter contains everything that a performance testing engineer needs to start performance testing a project.

Chapter 12 shows you how to generate a performance dashboard while executing test scripts.

Chapters 13 and 14 help in setting up JMeter and sample web applications to run the test scripts.

Architects, engineers, and quality assurance professionals will greatly benefit by reading this book. Project managers or other non-technical team members may want to glance through the book and read Chapter 2, “Performance Testing Primer,” to gain some understanding on performance testing in general.

We have developed an e-commerce web application for a hypothetical company called Digital Toys Inc., for the express purpose of illustrating the example test scripts in this book. Chapter 14 explains how to set up this sample application.

Test scripts developed in this book are hosted on GitHub. Any source code or other supplementary material referenced by the authors in this book is available to readers on GitHub via the book’s product page, located at www.apress.com/978-1-4842-2960-6. For more detailed information, visit <http://www.apress.com/source-code>.

For any quires or valuable feedback, feel free to get in touch with authors over e-mail:

Sai Matam at saimatam@yahoo.com.

Jagdeep Jain at jagdeep.jain@gmail.com.

CHAPTER 1



Foundation

In many companies, performance testing is not a priority until it becomes critical. The engineering team is then asked to complete performance testing in an extremely short time. Most of us are familiar with this situation and can relate to it. The engineering team needs to come up to speed with performance testing in general and with performance testing tools in a very short time.

In spite of realizing how critical performance testing is, companies often don't have an adequate budget for the needed commercial performance testing tools. Apache JMeter (hereinafter referred to as "JMeter") is the leading tool, and thankfully it is open source and thus is free.

While there is plenty of reference material on the Internet, there is no comprehensive guide to take you through all the steps of creating, running, and interpreting the results of a performance test using JMeter. This book aims to address this need.

This book discusses the basics and presents a framework for performance testing. You will be able to create a performance test plan that is based on the requirements. This book follows a step-by-step approach and guides you through the installation, configuration, test plan creation, execution, and result interpretation using JMeter.

Why Performance Testing?

Why bother to test the runtime performance of a web site? The answer is much more important than you realize. Improved web site performance directly translates to the bottom line (profitability) of a business, as it provides a better user experience, builds the brand, and retains customers. The improved performance uses fewer resources and utilizes them with more efficiency.

The current generation of web users is very demanding and discerning. They have short attention spans. They want everything quick. Twenty years ago a study revealed that a web site should load within six seconds or the user would click elsewhere! This was 20 years ago. Imagine what that attention time would be now? Four seconds? *Financial Times* (FT) conducted a performance test that clearly showed that a slow web site resulted in a bad user experience and negative financial impact. Even a delay of one second, over a long period of time, can have a huge impact on revenues.¹

¹Chadburn, Matt, Lahav, Gadi; "A Faster FT.com." Engine Room, Blog by the Financial Times Technology Department, April 6, 2016, <http://engineerroom.ft.com/2016/04/04/a-faster-ft-com/>.

Why JMeter?

JMeter is an industrial-strength performance testing tool. It is a top-level Apache open source project. But being free is not the only reason for its popularity. It's a very capable tool. This mature tool has been continuously revised since its inception in 2001.

The commercial tools may have fancier user interfaces or better-looking reports, but JMeter can be configured to provide the same capabilities. JMeter can be run in “distributed mode” in the cloud, generating the load of thousands of users. JMeter comes with an Apache License, which does not have any restrictions on usage, distribution, or modification.

JMeter has a standard format for writing the performance test. Most of the commercial tools support importing/exporting to the JMX (the JMeter test file format).

CHAPTER 2



Performance Testing Primer

This chapter discusses the overview of performance testing, performance criteria, types of performance tests, performance test infrastructure, and performance test strategy. This chapter does not refer to JMeter itself but to performance testing in general.

At the end of this chapter, you should have a good idea of how to structure your own performance tests and should be able to work toward a comprehensive performance testing strategy. Those who are already familiar with performance testing can proceed to the next chapter.

Performance Testing

Performance testing is the testing performed on a system or application to measure some of its attributes such as response time, throughput, scalability, etc. These attributes are called the performance criteria.

There are many types of performance tests, each of which requires different performance criteria to be measured. Depending on the nature of the application, some of the performance criteria may be more important than others.

Performance testing is different than functional testing. In performance testing, we are primarily focused on speed, whereas functional testing is concerned with correctness of the application behavior. Although performance tests are a part of the continuous build cycle along with the functional tests, the results of performance testing should be interpreted only after the application passes the functional tests.

Response Time

The time taken by the application to respond to the user's request is called the *response time*. It is measured in seconds or milliseconds as appropriate for the application. Every application should strive to minimize the response time. Figure 2-1 shows a typical web application request and response from the browser to the server (where the web application is hosted) and from the server to the browser.

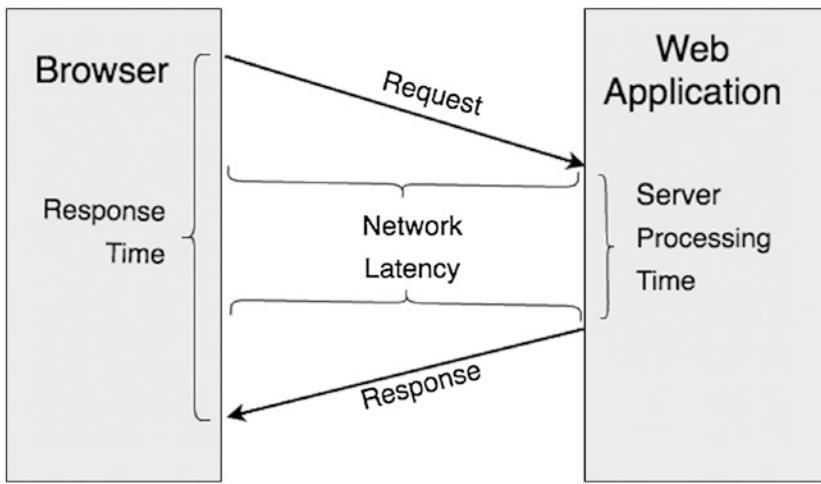


Figure 2-1. HTTP request-response

Some of the terms related to response time include the following:

- **Absolute Response Time:** This is the total time taken from the instant a user clicks on a link or submits a form until the response from the server is rendered completely.
- **Perceived Response Time:** The absolute response time for some complex web pages may be unacceptably high. To solve this issue, the request/response is structured into a set of request/responses that could be rendered progressively. This allows the users to continue reading or filling up a form, creating the perception that the response was fast even though the absolute response time is roughly the same. Perceived response time is the response time as perceived by the users.
- **Server Processing Time:** This is the time taken by the server to process the given input and generate the output. This can vary depending on the complexity of the request, server hardware, and the system load.
- **Rendering Time:** This is the time taken by the browser to parse and render the response received from the server. This depends on the complexity of the web page, presence of multimedia, presence of JavaScript, and whether the browser needs to generate content. It also depends on the system load on the user's device.
- **Network Latency:** This is the round-trip time taken by a data packet to travel over the network the server and back. This does not include the server processing time or the rendering time. This can vary widely depending on the location of the user, time of the day, and the network load.

Throughput

A sequence of request/responses constitutes a transaction. The number of transactions per unit time is called the *throughput*. It is measured in transactions/second or bandwidth (bytes/second). It depends on server hardware, system load, and network latency. The application should strive to maximize the throughput.

Utilization

Utilization is the ratio of the throughput of the application relative to its maximum capacity. This is a measure of how well the application is being used.

It is not desirable to operate above 80% utilization because the user requests do not arrive evenly, and the system should be able to handle a spike in the load.

Robustness

This is a measure of how well the application detects and handles various errors and exceptions. If the system crashes or becomes unavailable, the company's brand and reputation are at stake. *Mean Time between Failures (MTBF)* is a metric that is often used for this purpose.

Scalability

Scalability measures how well the system can expand its capacity when additional resources are added. Ideally the system capacity will increase linearly as additional resources are added. However, this is rarely achieved in practice. It's a good measure to know the resources that would be needed so that the system can handle the projected future load.

Vertical scalability is achieved by upgrading the hardware. For example, by adding more memory, disk, a better CPU, or additional CPUs.

Horizontal scalability is achieved by adding servers to the cluster. For example, by adding more web servers and application servers to a webfarm/cluster.

User Perception

The user is the ultimate judge of performance. The user has a set expectation regarding response time and robustness. If this is a casual news web site, the response time of six seconds may be okay. However, in the case of stock trading application, a sub-second response may be needed. Performance tuning is done if the measured performance is less than expected for that category of application. As mentioned earlier, the request/response may be restructured to improve the user perceived response time.

Cost

The system should consume fewer resources and save money for the company.

Utilization and throughput are some of the measures that influence cost.

Types of Performance Tests

The term *performance test* is used loosely to mean many different types of tests. Furthermore, the distinction between the various flavors of the performance tests is not widely understood. This is all the more reason for you to include a definition of these tests in your Performance Strategy document to make things clear.

Stress Tests

A *stress test* is a kind of performance test that tests the application beyond the normal limits. The application is subjected to excess load and after that its stability and performance are noted. This type of test is used to determine how the application responds to load spikes.

■ **Note** Performance tests evaluate and measure the application under a normal expected load. The stress test subjects the application to loads which are in far excess of normal.

Load Tests

A *load test* is a kind of performance test that's performed at the specified load level. So ideally, we would like to perform load tests at varying load levels to note the behavior of the application.

Peak Load Tests

A *peak load test* is performed at the load that the application is expected to handle. For example, e-commerce web sites experience their peak traffic during Black Friday, Cyber Monday, and the Christmas holidays. So a peak load test in this case would test the application within the load specification but at the higher end.

■ **Note** Stress tests test beyond the peak load.

Soak Tests or Endurance Tests

In a *soak test* (also called an endurance test), the application is subjected to a specified load that is within the specified limit but for a long duration. It is performed for many hours at a time. This test determines if the application is properly reusing its resources.

This test will surface problems like the following:

- Memory leaks in the application
- Database connections exhaustion
- Network connection exhaustion
- Log files becoming full and log rotation
- Other resource exhaustion

Scalability Tests

Successful web applications experience massive and sometimes exponential growth. So it is wise to measure how the application scales. Scalability is defined as how well the application handles the increase in load while still meeting the desired performance criteria.

A scalability test would increase the resources and test whether or not the application is providing a corresponding increase in capacity. Ideally, we expect linear scalability (i.e., doubling the hardware resources should result in double the application capacity).

Capacity Tests

A *capacity test* is a load test that establishes the maximum load that the application can handle while meeting the desired performance criteria. The resulting metric is called the *maximum capacity*. It is used in scaling the application and to estimate costs for future growth.

Spike Tests and Burst Capacity

A *spike test* is a load test where the application is subjected to brief periods of sudden increment in load, a small fraction beyond the maximum capacity. It is usually done to estimate the weakness/strength of an application. The application is expected to be robust and continue to meet the performance criteria during the spike. This metric is called the *burst capacity*.

Performance Smoke Tests

In a *performance smoke test*, a few common and essential use-cases along with use-cases pertaining to the code subject to change are together tested for performance. It is only when the smoke test succeeds that the full suite of performance tests are conducted. If the smoke test fails, no further performance tests are conducted until the performance defect has been rectified.

High Availability Test/Fail-Over Tests

Modern web application infrastructure is designed to be highly available and resilient to hardware and software failures. Ideally, the architecture should ensure that there is no single point of failure and that there are standby servers that can transparently take over without impacting the user experience.

In this test various equipment and software failures are simulated and relevant performance tests are run to verify that the application is still meeting the performance criteria.

SIMIAN ARMY AT NETFLIX INC.¹

Netflix is a well known provider of digital media streaming services, serving more than 74 million subscribers in 190 countries. Performance and robustness are critical for it to survive.

The Netflix engineering team has come up with the concept of “chaos engineering,” in which defects are deliberately introduced to the production system to test and validate if the system is robust enough to fail-over and recover from the errors while meeting the pre-established performance criteria. It developed a set of tools dubbed as the *Simian Army*, a couple of which are summarized here.

Chaos Monkey

It randomly disables the production instances to make sure that the system can survive this common type of failure without any customer impact.

Chaos Gorilla

Similar to Chaos Monkey, but simulates an outage of an entire Amazon availability zone. It verifies that the services automatically rebalance to the functional availability zones without user-visible impact or manual intervention.

¹“Chaos Engineering Upgraded,” The Netflix Tech Blog, September 24, 2015, <http://techblog.netflix.com/2015/09/chaos-engineering-upgraded.html>.

Netflix Chaos Kong in Operation

It is very rare that an AWS Region becomes unavailable, but it does happen. On September 20th, 2015, Amazon's DynamoDB service experienced an availability issue in their US-EAST-1 region. That instability caused more than 20 additional AWS services that were dependent on DynamoDB to fail. Some of the Internet's biggest sites and applications were intermittently unavailable during a six- to eight-hour window that day. This had minimal impact on Netflix, thanks to Chaos Kong.

Netflix did experience a brief availability blip in the affected region, but it sidestepped any significant impact because Chaos Kong exercises prepared them for incidents like this. By running tests on a regular basis that simulate a region-level outage, they were able to identify any systemic weaknesses early and fix them. When US-EAST-1 actually became unavailable, their system was already robust enough to handle a traffic fail-over.

Figure 2-2 shows a chart of Netflix's video play metrics during a Chaos Kong exercise. These are three views of the same eight-hour window. The top view shows the aggregate metric, while the bottom two show the same metric for the West region and the East region, respectively.

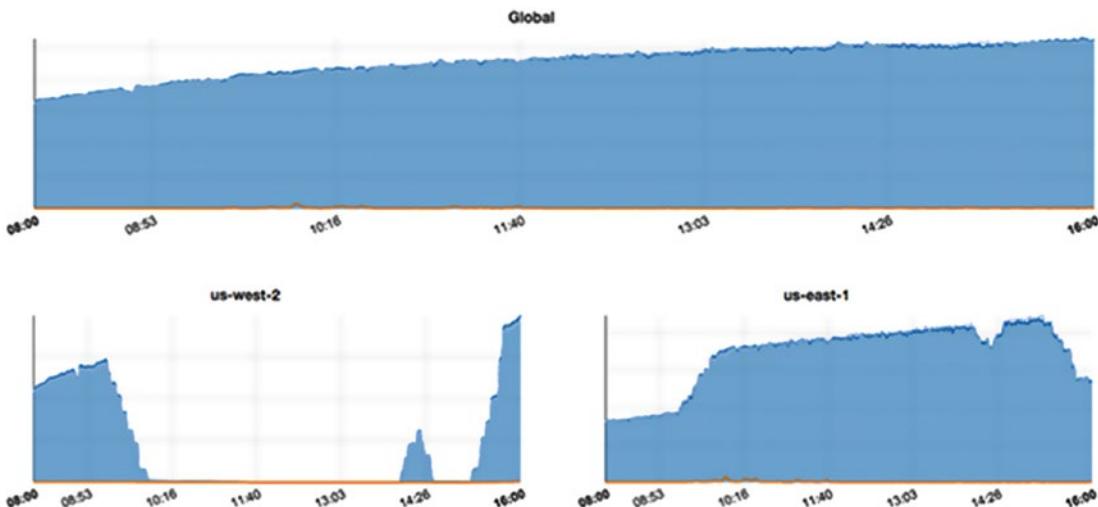


Figure 2-2. Netflix Chaos Kong in operation

In the bottom row in Figure 2-2, you can clearly see traffic evacuate from the West region.

The East region gets a corresponding bump in traffic as it steps up to play the role of savior. The aggregate metric shown in the top chart does not show any impact, demonstrating that the system was resilient to the failover. At the end of the exercise, the traffic reverted to the West region. The aggregate view shows that their members did not experience any adverse effects. Netflix runs Chaos Kong exercises like this on a regular basis; it gives them confidence that even if an entire region goes down, they can still serve their customers.

The Performance Test Environment

The hardware and software used to conduct performance testing is called the *performance test environment*. It is usually separate from the production environment. Although not advisable, sometimes performance testing is done in the production environment itself.

The Need for Separate Performance Environment

Performance testing should not be performed in the production environment as it may negatively impact the user experience.

The load from performance testing may:

- Crash the system
- Degrade the application response time
- Create security holes due to the use of test accounts
- Litter the production databases with performance test input and output data
- Fill up the databases, application log files, and system log files
- Affect the analytics

In the long run, it is wise to have a dedicated performance environment as it would avoid these pitfalls and would provide the benefit of *Continuous Integration* (CI). In fact, some companies have more than one. The advent of cloud computing enables such environments to be provisioned on demand at a low cost.

Performance testing needs to happen before software is deployed into production. Ideally this needs to be a part of the build process so that any performance defects introduced as a result of software changes can quickly be detected and rectified.

Despite the benefits of a performance test environment, sometimes performance testing is done in the production environment because it is prohibitively expensive to duplicate the production environment in terms of hardware, additional software licenses, and third-party services.

The Performance Environment Should Be Like the Production Environment

The performance environment should be modeled after the production environment. In an ideal scenario, the performance environment should be a replica of the production environment in terms of hardware, software, network, components, and topology. However, due to budget and other constraints, this may not be possible. In such cases, the performance environment should mimic the production environment in all key aspects. Some logical compromises will have to be made. For example, instead of a cluster of four web servers, the team may choose to go with a minimal cluster of two servers. They may decide to go with a single instance of the database instead of a master-slave database configuration in production.

The Performance Environment Should Be Isolated

The performance environment should be on a different subnet, isolated from the production environment so that any activity on the performance subnet will not influence production and vice versa.

Performance Testing Tools

Performance testing tools should address load generation, performance data collection, and analysis and reporting. Tools should be identified and their specific features/usage explored and documented well in advance. These tools need to be installed on relevant servers in the environment.

The Performance Testing Strategy Document

The performance testing strategy is a document that defines performance requirements and goals, as well as the approach to achieve and maintain them. It contains the performance testing policy and methodology. This document captures the thinking of the accountable business and IT executives and is signed off by them.

Performance Requirements

As detailed, there are many performance criteria that are used in the industry. While all of them could be useful, only a subset is applicable to your specific application. Further, an even smaller number may be of critical importance due to contractual obligations, Service Level Agreements (SLAs), or the application owner's stipulations. The criteria thus identified are called *performance requirements*.

Sometimes the nature of the business dictates these requirements. For example, for a stock trading application, a sub-second response time is a requirement.

Every release has to meet these requirements.

Performance Goals

Performance goals are criteria that are desired but not critical. These are often determined by the performance of similar applications from competitors. Meeting these goals would be to the advantage of the business. For example, for a news web site, a response time of six seconds may be desirable.

Note Performance requirements are absolute, whereas performance goals are aspirational.

Performance Test Suite

The *performance test suite* is a set of tests that measures the performance requirements and goals. The company's performance testing strategy may call for multiple performance test suites. Not all the test suites need to be executed each time. For example, a test suite for scalability would be run in preparation of busy times such as Christmas, while a smoke test suite is executed as a part of the build process.

Application usage patterns can be identified either by analyzing the web application usage logs or the usage patterns are provided by the product owner or marketing. These are taken into account while creating the test suite.

The test suite undergoes changes to meet the changing needs of the application. New performance tests are added to address new product features; old performance tests are deprecated for obsolete features; existing performance tests are enhanced based on the analysis of performance reports, the feedback from the product owner, or data from the marketing department.

Performance Reporting and Analysis

Performance reports are analyzed to detect instances where performance requirements are not being met; these are logged as critical defects.

The performance is also compared to the baseline to note the trends. Any deviation is brought to the attention of the team for further analysis and action.

Performance Tuning

The engineering team reviews and modifies the application to address defects and concerns raised by the performance reports. These tuning changes may include modifications to configuration, code, network, architecture, topology, etc. Figure 2-3 shows the performance testing/tuning process flow.

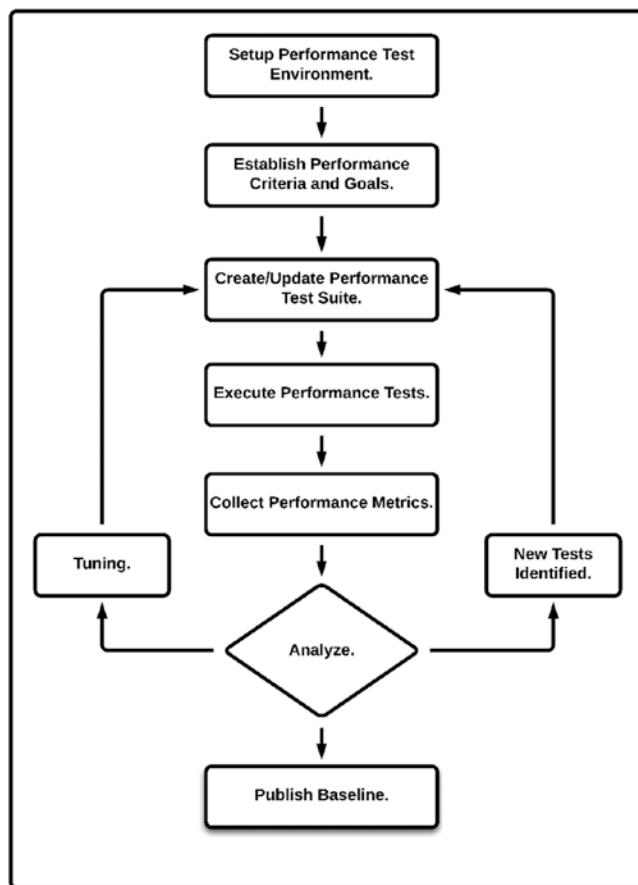


Figure 2-3. Performance testing process

Conclusion

In this chapter, you learned the basic terms of performance testing, types of performance tests, performance test environment, and performance test suites. These are helpful in developing a performance testing strategy. The next chapter starts with JMeter and explains the various components of JMeter. You will be developing and running a JMeter test script and will also learn various ways of starting/stopping/shutting down a JMeter test script.

CHAPTER 3



Your First JMeter Test

This chapter discusses the various components of JMeter and helps you develop the first JMeter test script. You will find out about the various ways (GUI/non-GUI) of running and stopping/shutting down JMeter test scripts in standalone test execution and server mode of test execution.

At the end of this chapter, you will have a good idea of the various components of JMeter and be able to write a simple JMeter test script, as well as start/stop/shut down tests from GUI and non-GUI modes.

Before starting with the JMeter test script development, let's walk through the components of a *test plan*.

Components of a JMeter Test

Caution JMeter test scripts have a filename extension of `.jmx`. This is not related to Java Management Extensions (JMX).

A JMeter test script consists of hierarchical and ordered components organized in the form of a tree.

Each of these components has properties that can be configured in the following ways:

- By `jmeter.properties` file
- By using command-line parameters
- By editing the `.jmx` file directly using a text editor
- By using the GUI
- By using values extracted from the responses received to sampler requests

Test Plan

The *test plan* is the top-most element and is the root of the tree. For a test plan, the name, description, and user variables can be configured.

Thread Group

Every test has one or more *thread groups*. A thread group is a child element of a test plan. Each thread group represents a use-case. As you will see later in the book, a thread group can be configured with the number of threads, ramp-up time, and other useful properties that allow you to control its behavior.

Controller

Each thread group has one or more controller elements. Logical *controllers* decide the flow of execution. They determine the next Sampler to execute. JMeter comes with many built-in logical controllers that provide precise control flow. For example, the If controller and Switch controller provide branching; the ForEach, While, and For provide iteration flow. There is a controller for every programming construct. A custom controller, if needed, could be developed using the plugin API mechanism provided by JMeter.

Sampler

Sampler is a child element of a thread group or a controller. It sends requests to the server. For every protocol, we need a separate sampler. Out of the box, JMeter comes with many samplers. For example, to send a HTTP request, you use a HTTP Request Sampler. Custom samplers can be developed using the JMeter plugin mechanism.

Listener

Listeners listen to the responses from the server and assemble and aggregate the performance metrics. They are used to display graphs. We need at least one listener per test script so that we can interpret and understand the results of the performance test.

Timer

A *timer* introduces a delay in the flow. Delay is needed between sampler requests for the following reasons:

- To simulate the time that the user takes to perform the next action on the web page
- To simulate a realistic load distribution on the server

Add timers as child elements of samplers or logic controller that need the delay.

Assertions

Assertions are used to verify that the server responses are as expected. Assertions test various status codes, and then pause, alert, or log bad request/responses. It is important to ensure that the server is not returning any error codes during the execution.

Adding an assertion as a child of the sampler restricts it to a single sampler. Otherwise, assertions will apply to all samplers that are in scope.

Add an Assertion Results Listener to the thread group to view the assertion results. Failed assertions will also be displayed in the View Results Tree and the View Results in Table Listeners, and will count toward the error percentage, for example, in the Aggregate Reports and Summary Reports.

Config Element

Configuration elements are placeholders for properties, including user-defined properties and variables. For example, the *HTTP Cookie Manager* is a configuration element. Configuration elements can be scoped out with a nesting level.

Pre-Processors

Pre-processors take the request and modify it (substitution, enhancement, dereferencing variables, etc.) before the sampler sends it to the server.

Post-Processors

Post-processors process the response from the server. They are used to process the server response and modify the component settings or to update variables.

Order of Component Execution

Programming languages evaluate expressions by following certain rules based on operator precedence. For example, in the expression $a + b * c$, the operation $b * c$ is performed first and the result is added to a . Although the operator $*$ is not the first operator appearing in the expression, it is applied first as it has a higher precedence value.

Similarly, JMeter follows certain rules while executing the components in a test plan. Within a thread group, the order of execution of the components follows the order shown in Figure 3-1. Even if a Listener component was placed as the first element in the thread group, it will be executed after samplers and post processors. However, controllers and samplers have the same precedence, so they will be executed in the order in which they appear in the test plan.

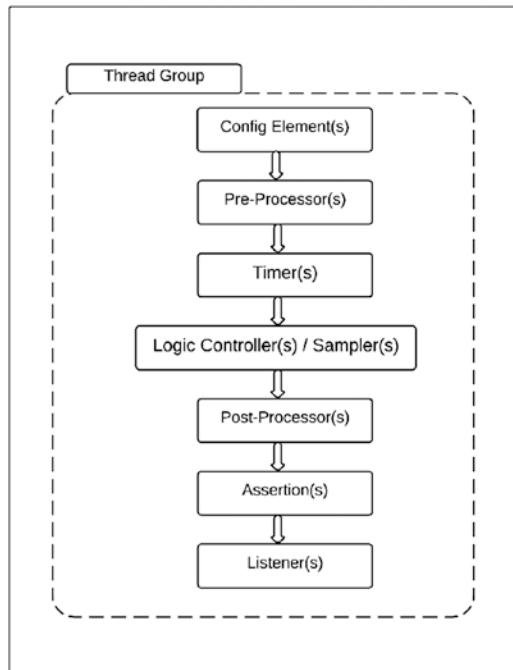


Figure 3-1. Order of execution

JMeter allows the elements to be nested. For example, a listener, a timer, and a sampler can be nested under a logical controller. In this case, when the logical controller is being executed, the order of execution of its child nodes would be the timer, the sampler, then the listener.

Figure 3-2 shows the order of execution in the test plan.

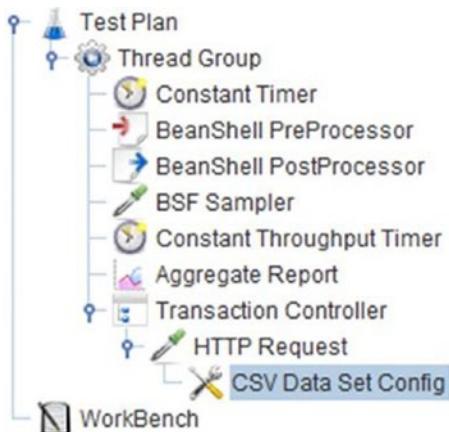


Figure 3-2. Test plan

As you can see in the JMeter test plan, the components are not in sequence.

However, JMeter will rearrange them according to the Order of Execution diagram shown in Figure 3-1. Note that the *CSV Data Set Config* is nested under the transaction controller. The child elements nested under the transaction controller would again follow the Order of Execution diagram and get sequenced.

For the given *test plan*, the execution order will be:

1. *BeanShell Pre-processor*
2. *Constant Timer*
3. *Constant Throughput Timer*
4. *Transaction Controller*
5. *CSV Data Set Config*
6. *HTTP Request*
7. *BeanShell PostProcessor*
8. *Aggregate Report*

Note All the examples mentioned in this book have been coded against a sample web application called Digital Toys Inc. Refer to Chapter 14 for setup instructions.

Simple JMeter Test

This first test script is very simple.

1. Simulate a user browsing the Digital Toys Inc. web application home page (<http://localhost:8080/dt>).
2. Check for an HTTP status code of 200.

Follow these steps or download `FirstTestPlan.jmx`.¹

1. Start JMeter GUI.
2. Create a test plan and give it a meaningful name, such as `First Test`.
3. Click on Test Plan and go to `Edit > Add > Threads (Users)` and add Thread Group. Configure the **Number of Threads (Users)** as 1 and **Loop Count** as 1 (see Figure 3-3).

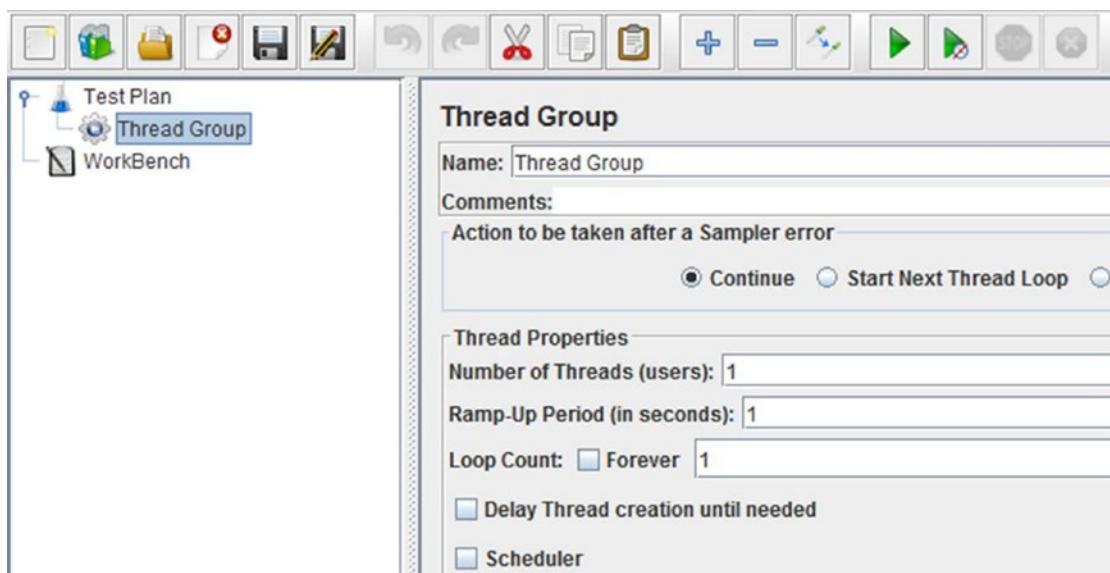


Figure 3-3. Thread group

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_03/FirstTestPlan.jmx

- Click on Thread Group and go to Edit ▶ Add ▶ Sampler and add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt. Uncheck **Follow Redirects** (see Figure 3-4).

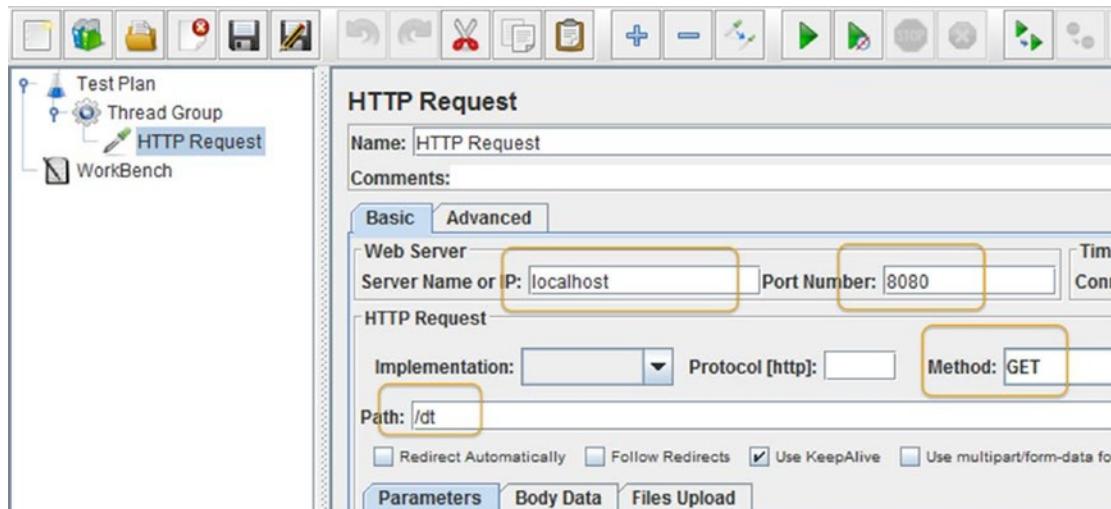


Figure 3-4. HTTP request

- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions and add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200 (see Figure 3-5).

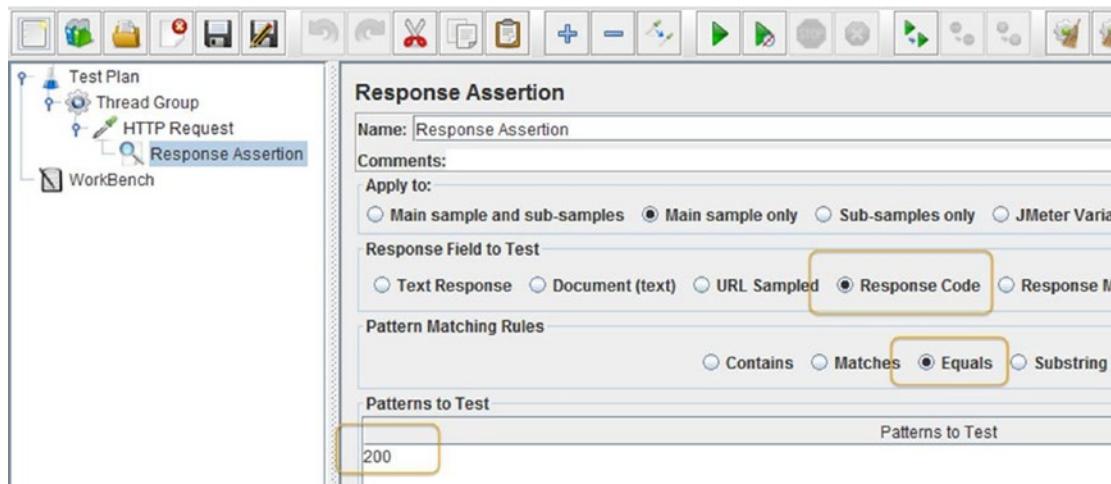


Figure 3-5. Assertions

6. Click on Thread Group and go to Edit ► Add ► *Listener* and add View Results Tree (see Figure 3-6).

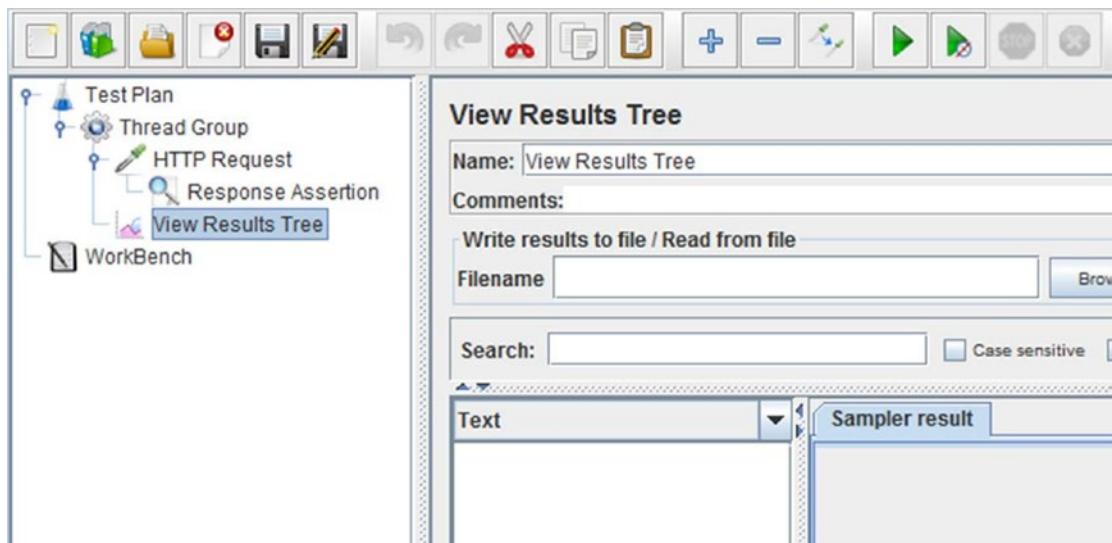


Figure 3-6. Run test

7. Save the test plan.
8. Go to Run ► Start to run the test (on Mac OSX, type CMD+R).
9. Verify the responses in the View Results Tree (see Figure 3-7).

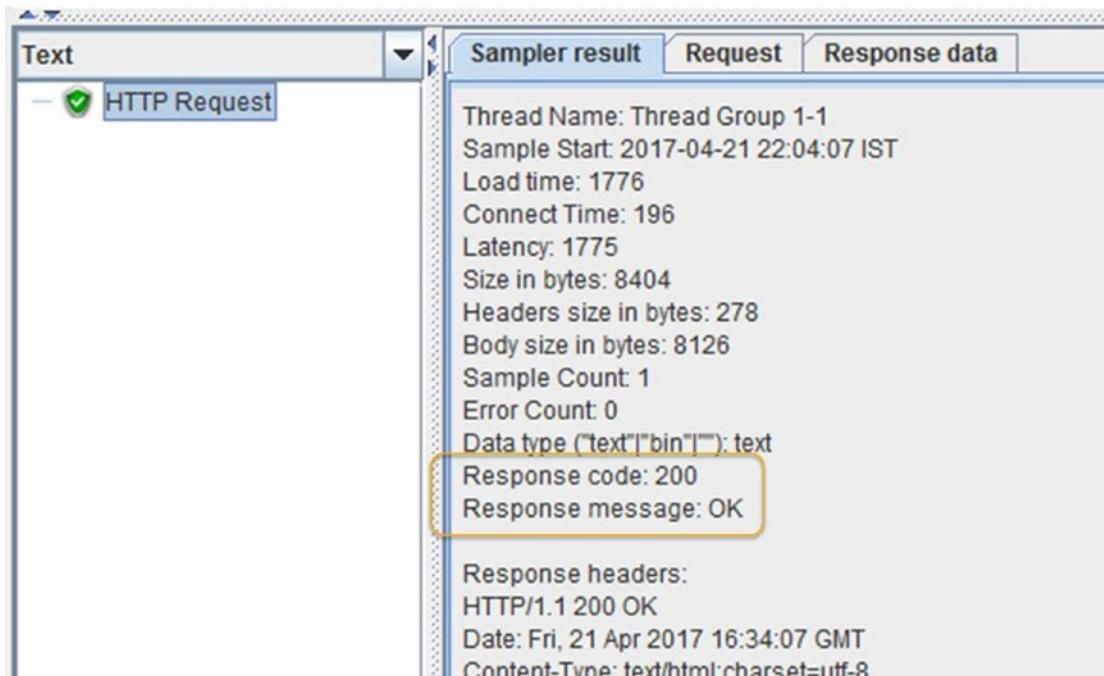


Figure 3-7. Test results

Now, let's simulate a load of 10 users by following these steps.

1. Highlight Thread Group and configure **Number of Threads (Users)** as 10, **Ramp-Up Period (in seconds)** as 0, and **Loop Count** as 1.
2. Save the test plan.
3. Go to Run ▶ Start to run the test (on Mac OSX, type CMD+R).
4. Verify the responses in the View Results Tree (see Figure 3-8).

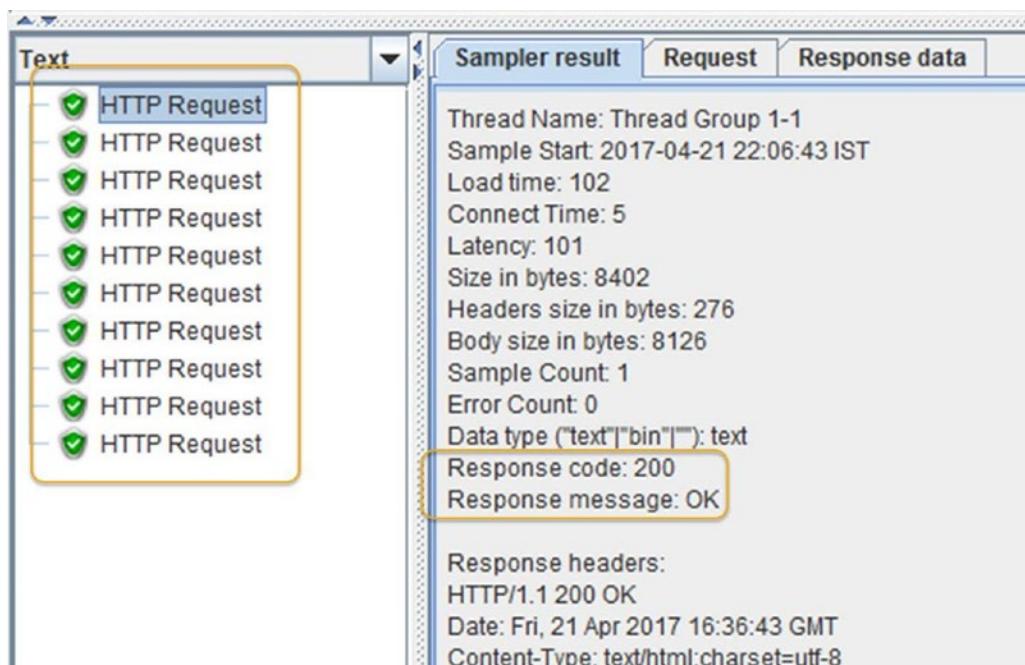


Figure 3-8. Execution results

Navigate to each response under the View Results Tree and verify the **response** and **response time** (**Load Time**). Calculate the average response time as the sum of the response time of all threads/number of threads.

GUI Mode

JMeter GUI mode provides several options to start/stop test(s) (see Figure 3-9).

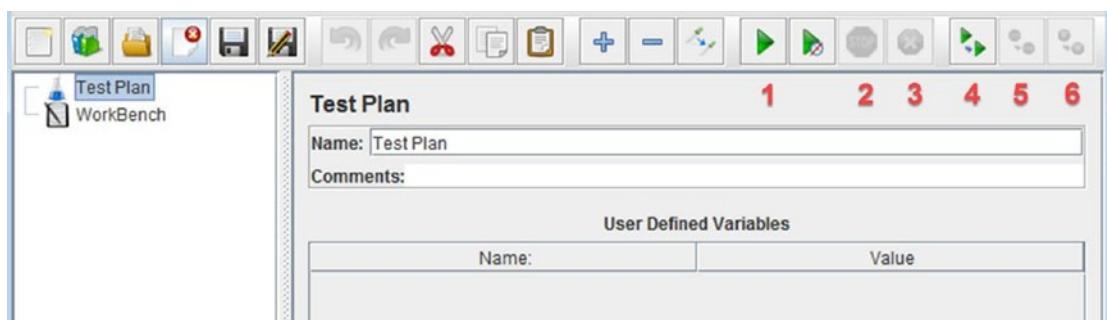


Figure 3-9. JMeter test execution options

- **Start:** Start a test on the local machine
 - **Stop:** Abruptly stop the test that is running on the local machine
 - **Shutdown:** Stop the test gracefully, allowing the threads to wind down
 - **Remote Start All:** Start the test on all the remote JMeter hosts
 - **Remote Stop All:** Abruptly stop the test on all the remote JMeter hosts
 - **Remote Shutdown All:** Gracefully shut down the test on all the remote JMeter hosts

Note Use the GUI to create and debug the tests scripts. Use the non-GUI mode to execute the test and collect the results.

Non-GUI Mode

GUI mode is not desirable because:

- It consumes a large amount of resources, thus interfering with the test results
 - It may not be available in some environments (such as a remote shell)

To avoid these issues, use the non-GUI mode.

To see the command line options provided by JMeter, issue the following command in the CMD prompt.

```
jmeter --help
```

This will show you the help options provided by JMeter.

```
C:\apache-jmeter-3.0\bin>jmeter --help  
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter.log
```

 3.0 r1743807

Copyright (c) 1999-2016 The Apache Software Foundation

To list all command line options, open a command prompt and type:

jmeter.bat(Windows)/jmeter.sh(Linux) -?

To run Apache JMeter in GUI mode, open a command prompt and type:

jmeter.bat(Windows)/jmeter.sh(Linux) [-p property-file]

To run Apache JMeter in NON_GUI mode:

Open a command prompt (or Unix shell) and type:

```
jmeter.bat(Windows)/jmeter.sh(Linux) -n -t test-file [-p property-file] [-l log-file]
```

To run Apache JMeter in NON_GUI mode and generate a report at end :

Open a command prompt (or Unix shell) and type:

```
jmeter.bat(Windows)/jmeter.sh(Linux) -n -t test-file [-p property-file] [-l log-file] -e -o [Path to output folder]
```

To generate a Report from existing CSV file:

Open a command prompt (or Unix shell) and type:

```
jmeter.bat(Windows)/jmeter.sh(Linux) -g [log-file] -o [path to output folder (empty or not existing)]
```

To tell Apache JMeter to use a proxy server:

Open a command prompt and type:

```
jmeter.bat(Windows)/jmeter.sh(Linux) -H [your.proxy.server] -P [your proxy server port]
```

To run Apache JMeter in server mode:

Open a command prompt and type:

```
jmeter-server.bat(Windows)/jmeter-server(Linux)
```

```
C:\apache-jmeter-3.0\bin>
```

Executing a Single Test

Let's explore command-line options using the following examples.

Download `FirstTestPlan.jmx`² and issue the following command in the CMD prompt.

```
C:\>jmeter -n -t FirstTestPlan.jmx -l test-run.jtl
```

Note that `test-run.jtl` is the test execution log file.

²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_03/FirstTestPlan.jmx

Proxy Server Setting

If you are behind a proxy server, append the following options.

```
-H <proxy host server name>, -P <port number>
```

Start JMeter in Server Mode

JMeter also runs in master-slave formation, known as server mode. To run JMeter in server mode, issue the following command in the CMD prompt.

```
C:\>jmeter-server
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter-server.log
Created remote object: UnicastServerRef [liveRef: [endpoint:[172.17.65.111:61425]
(local),objID:[-10c0a943:15c299a388a:-7
fff, -3710696168706880958]]]
```

Stop/Shutdown JMeter

To stop JMeter test execution, issue the following command in the CMD prompt. Once the command executes, it sends a stop signal to the JMeter test. The response message in the CMD prompt should read `StopTestNow request to port 4445`.

```
C:\apache-jmeter-3.0\bin>stoptest.cmd
Sending StopTestNow request to port 4445
```

To shut down JMeter, issue the following command in the CMD prompt. Once the command executes, it will send a shutdown signal to the JMeter test. The response message in the CMD prompt should read `Shutdown request to port 4445`.

```
C:\apache-jmeter-3.0\bin>shutdown.cmd
Sending Shutdown request to port 4445
```

Conclusion

In this chapter, you learned about the components of a JMeter test and wrote a simple JMeter test. You also learned how to start/stop/shut down JMeter test execution via GUI mode and from the command prompt. In the next chapter, you will learn to record user actions and create JMeter test scripts by using *HTTP(s) Test Script Recorder*.

CHAPTER 4



JMeter Test Script Recorder

This chapter discusses how to record a JMeter test script by using HTTP(S) Test Script Recorder. We cover the WorkBench and Recording Controller, how to configure a proxy port for recording HTTP(S) calls, and inclusion and exclusion of specific URL patterns. You will also see how to record an example using the Digital Toys Inc. web application.

At the end of this chapter, you will have a good idea of recording user actions via a browser and will be able to develop a simple JMeter test script. Those who are already familiar with the HTTP(S) Test Script Recorder can proceed to the next chapter.

Before starting with the *HTTP(S) Test Script Recorder*, you need to make sure that JMeter understands browser actions. You need to configure the browser and use JMeter as a proxy server.

Once you complete the configuration, you can use the browser and perform the use-case specific steps on a browser. JMeter, being in the middle, can intercept the requests from the browser, record them, and forward them to the server. Similarly, JMeter can record the web application responses before forwarding them back to the browser. See Figure 4-1.

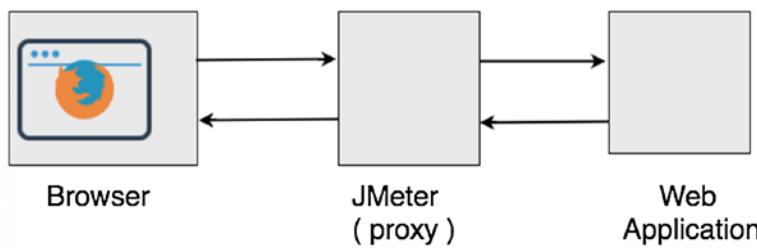


Figure 4-1. JMeter as a proxy server

JMeter WorkBench

JMeter's *WorkBench* provides a temporary workspace to store test elements, including a thread group. When the JMeter GUI starts, it is pre-populated with an empty test plan and an empty WorkBench. When JMeter is configured as a proxy, it can record the browser activity in the WorkBench. Users can then copy/paste recorded requests from the WorkBench into the test plan.

Test elements in the WorkBench are considered a work-in-progress. They are not saved with the test plan unless you check the Save WorkBench check-box. See Figure 4-2.

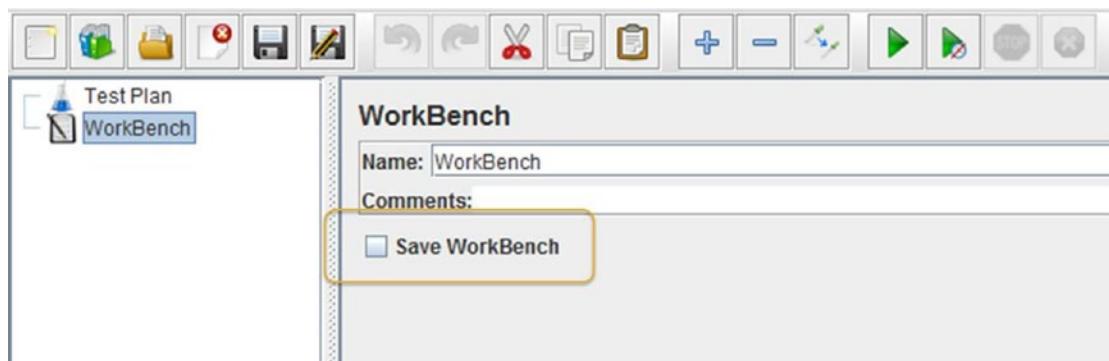


Figure 4-2. Save WorkBench

Apart from the test elements, it can contain *non-test elements* like a HTTP(s) Test Script Recorder. See Figure 4-3.

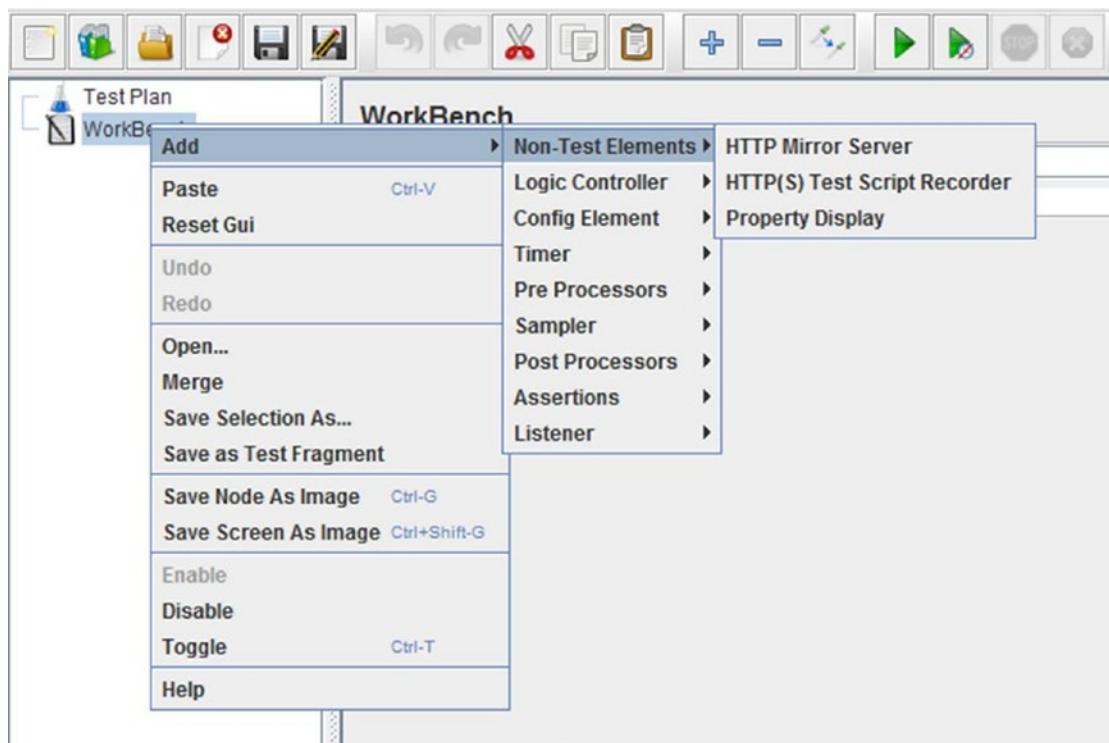


Figure 4-3. WorkBench options

WorkBench provides us with a few configuration options for recording test scripts, such as global settings, test plan content, *URL patterns to include*, *URL patterns to exclude*, and *notify child listeners of filtered samplers*. See Figure 4-4.

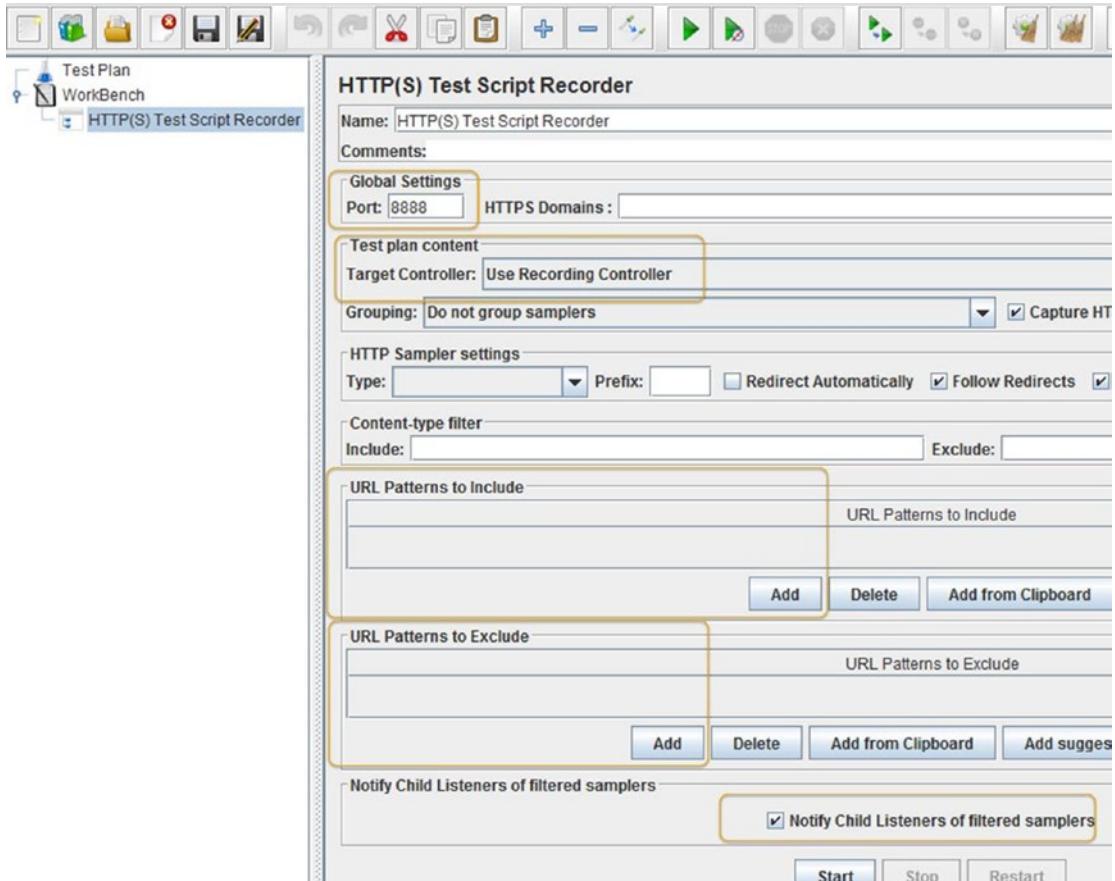


Figure 4-4. WorkBench

In Global Settings, the Port configuration is used to configure JMeter to listen on a specified port.

Test plan content allows you to specify the controller where the proxy will store the generated samples. It is set to Recording Controller by default.

The URL Pattern to Include option allows you to add a regular expression that specifies the URLs to include. For example, use the regular expression `./product\//` to specify a URL containing the product.

The URL Patterns to Exclude option allows you to add a regular expression to specify the URLs to exclude. For example, you can use the regular expression `.\.(css|js)` to indicate that you don't want CSS or js.

Selecting the Notify Child Listeners of Filtered Samplers check box causes the child Listener (for example, the View Results Tree) not to see these requests.

JMeter Recording Controller

By using Recording Controller in the WorkBench, you can capture the recording of the user actions performed in the browser.

After recording the test via the WorkBench, the user can run the test to check if it is working as expected.

Browser Proxy Settings

We need to set the JMeter Port setting in Global Settings and the browser proxy port to the same value: --7070.

Let's set up the proxy in the Firefox browser. See Figure 4-5.

1. Open Firefox browser and choose Preferences > Advanced > Network > Connection Settings > Manual Proxy Configuration. Configure **HTTP Proxy** as localhost and **Port** as 7070.
2. Select **Use This Proxy Server for All Protocols**.
3. Clear the **No proxy For** text area.
4. Click OK.

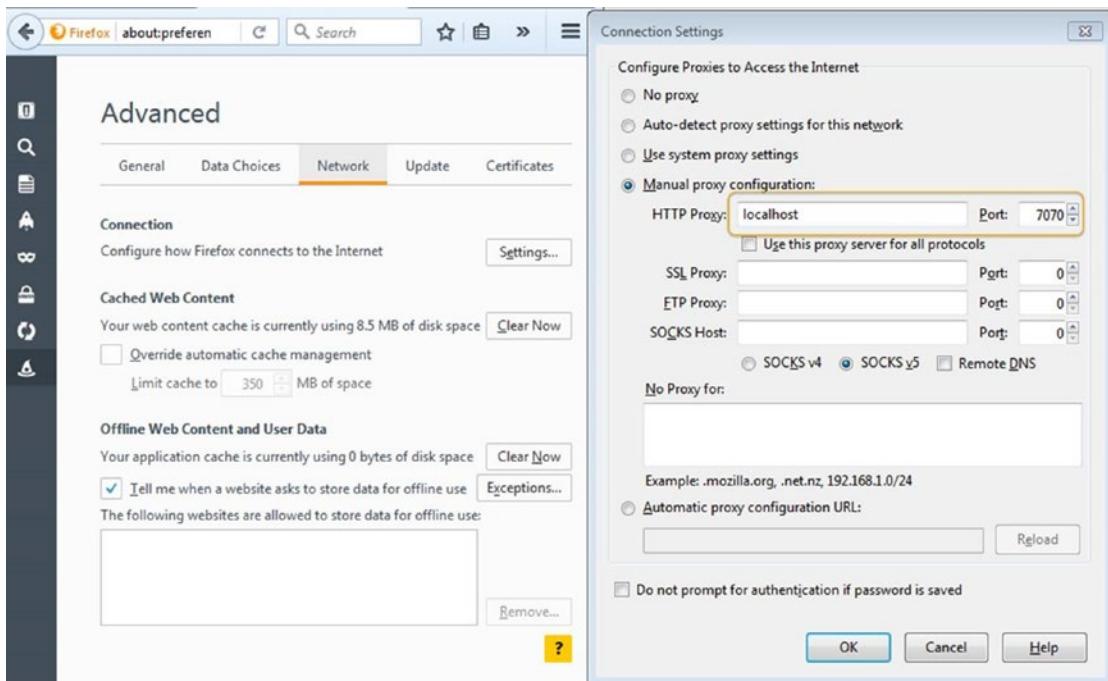


Figure 4-5. Firefox proxy settings

JMeter is now the proxy between the Firefox browser and the web server.

Recording Example

Using the Digital Toys Inc. web application, we will now place an order. Because JMeter has been set up as the proxy, you can record all the user interactions on the browser.

Follow these steps.

1. Sign in (user: user1@dt.com, password: user1).
2. Go to Check Detail ► Add To Cart ► Checkout ► Add Billing/Shipping Address ► Add Credit Card ► Place Order ► Sign Out.

Let's illustrate the process of developing a test script by recording the user actions.

Follow these steps or download [FirstRecordingTestPlan.jmx](#)¹:

1. Create a test plan and give it a meaningful name, such as First Recording Test.
2. Click on WorkBench and go to Edit ► Add ► Non-Test Elements. Then add HTTP(S) Script Test Recorder. In Global Settings, configure Port as 7070. Select Target Controller as WorkBench > HTTP(S) Test Script Recorder.
3. Click on the Start button. JMeter will show a Root CA Certificate dialog box. Click OK.
4. Perform the use-case actions in the browser; they will be shown as in Figure 4-6.

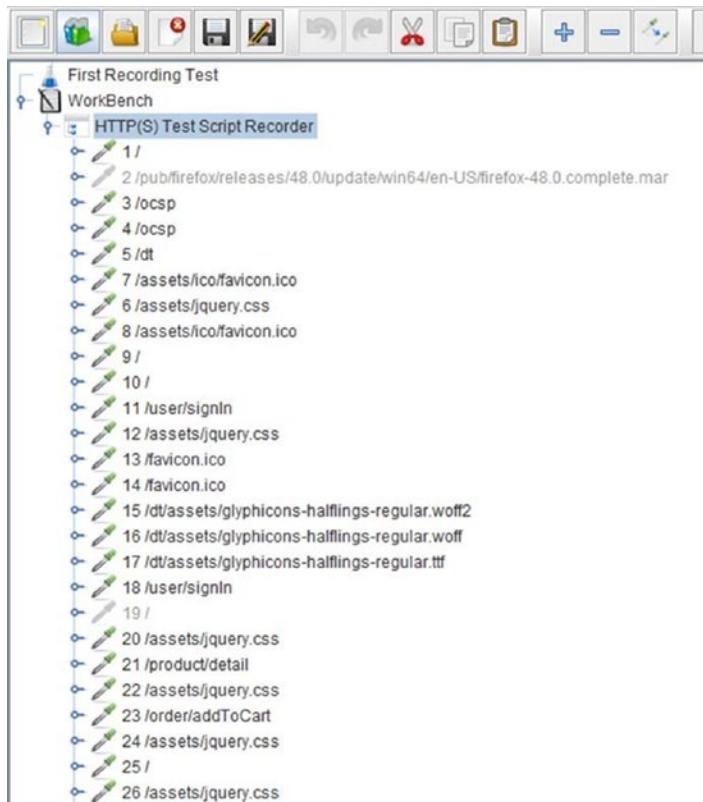


Figure 4-6. Recording browser actions

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_04/FirstRecordingTestPlan.jmx

5. Click on the Stop button.
6. After recording, you will notice requests for .js, .css, and other resources that are not needed. You can exclude these requests by specifying Exclude Regular Expression. Click on Add Suggested Excludes, and this action will add a sample expression. Modify this expression, as shown in Figure 4-7.

.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*

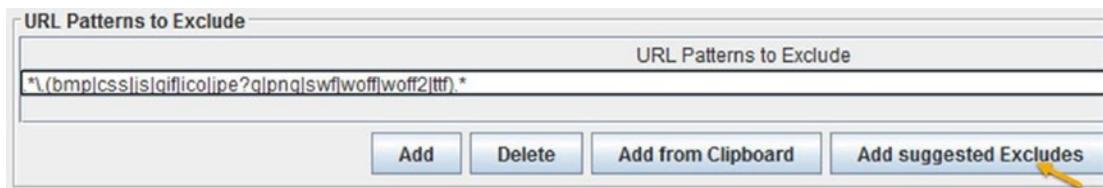


Figure 4-7. Exclude regular expression

Note We have excluded .bmp, .css, .js, .gif, .ico, .jpeg, .png, .swf, .woff, .woff2, and .ttf requests. This is because our focus was to test the performance of the dynamic responses from the server.

7. Clear the previous recording.

You will now re-record the use-case without the unwanted requests.

1. Click on the Start button. JMeter will show a dialog box. Click OK.
2. Perform the use-case actions in the browser.
3. You will now see that the recorded calls have been filtered for unwanted resources. See Figure 4-8.

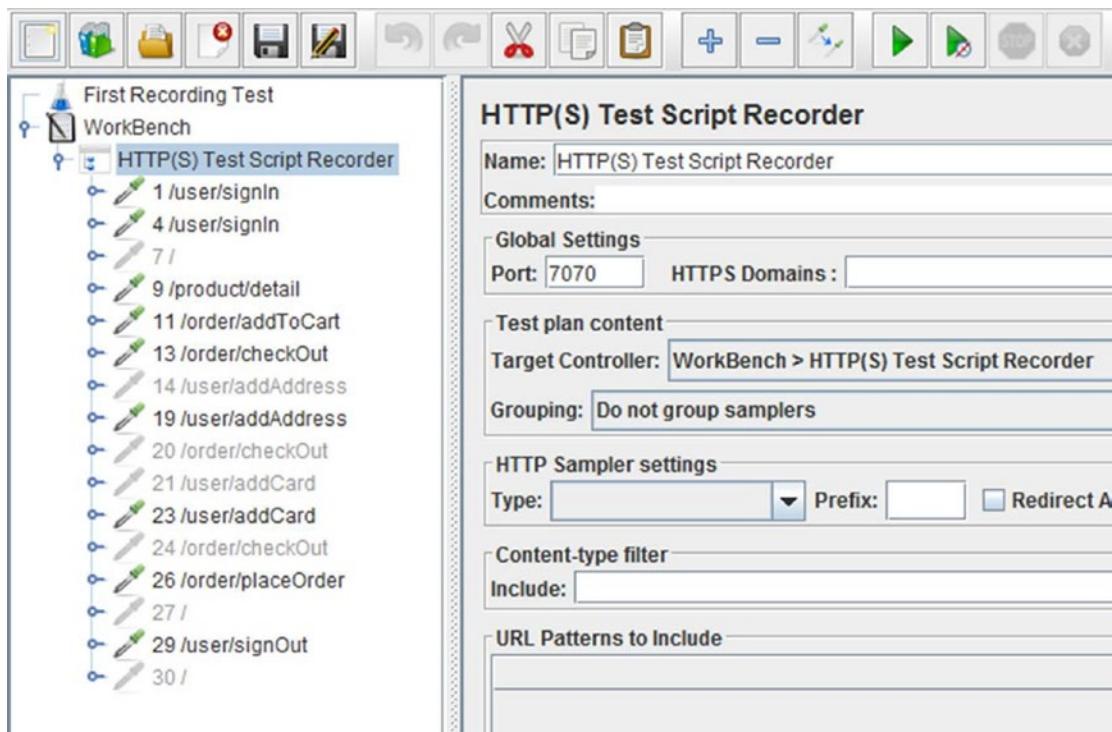
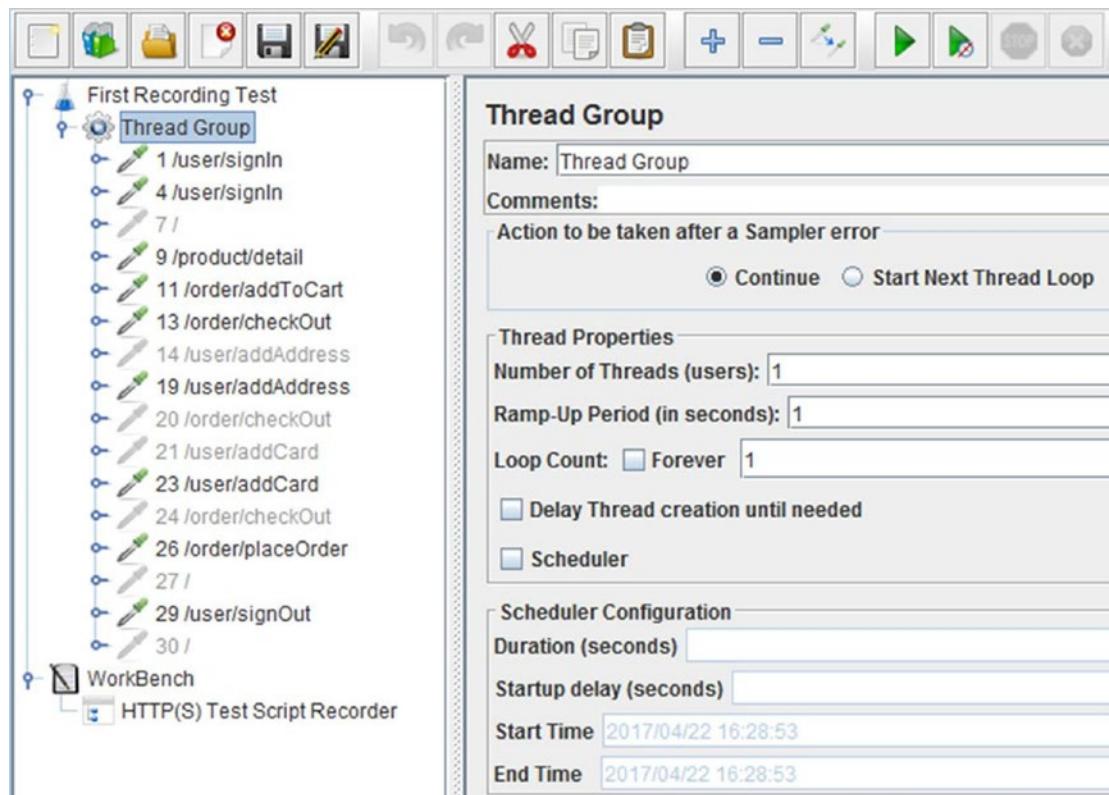


Figure 4-8. Recording browser actions

4. Click on Test Plan and go to Edit ► Add ► Threads (Users). Add a thread group.
5. Select all recorded browser actions from WorkBench and then drag and add them as child elements of the thread group. The result will look as shown in Figure 4-9.

**Figure 4-9.** Final recording

6. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
7. Save the test plan.

Note Each of the requests in the HTTP(s) Test Script Recorder starts with a request number. However, these numbers may not be in sequence, as some requests may have been discarded based on the URL exclusion/inclusion filters.

To verify that the test is working correctly, follow these steps.

1. Click on Test Plan and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
2. Run the test. Results will be similar to those shown in Figure 4-10.

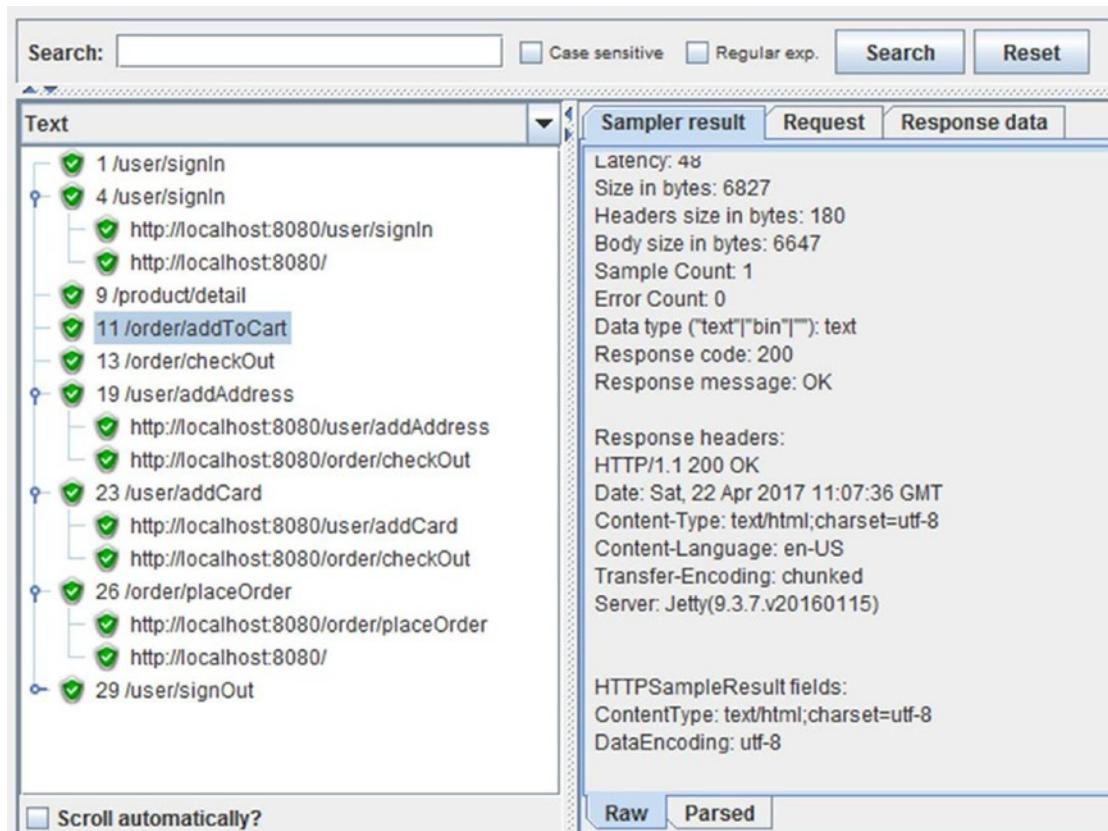


Figure 4-10. View results

After the test executes, open the browser, navigate to the Digital Toys Inc. web application, log in with the user1@dt.com username and user1 password, then navigate to Order History and check if the order is present. You should see two orders—the first order was created while recording the test and the second order was created due to the test execution. See Figure 4-11.

The screenshot shows a web page titled "Digital Toys" with a dark header bar. Below the header, the title "Your Order History" is displayed in blue. Underneath, it shows "Order Date: 2017-04-22" and "Status: Order Placed and Shipped". A table follows, listing one item: "Good Shot Camera" at \$125.95, with a "Total" row also showing \$125.95. Another identical section below it also lists the same order details and table. At the bottom right, there is a blue button labeled "Continue Shopping".

S. No.	Product Name	Price
1.	Good Shot Camera	\$ 125.95
	Total	\$ 125.95

S. No.	Product Name	Price
1.	Good Shot Camera	\$ 125.95
	Total	\$ 125.95

[Continue Shopping](#)

Copyright© 2016 - JMeter By Example - All rights reserved.

Figure 4-11. Order history

Note We excluded .js, .css, .ico, .ttf, .woff, etc. requests, which are basically browser-based calls instead of server calls. This is because the focus here is to test the performance of dynamic responses from the server.

Conclusion

In this chapter, you learned to configure JMeter as a proxy and use the HTTP(S) Test Script Recorder. You used regular expressions to include/exclude specific URLs in your test script and successfully developed a JMeter test script. In the next chapter, you will learn about the test plan and its components and explore various options for each of those components.

CHAPTER 5



JMeter Test Plan Components

This chapter discusses various components of a typical JMeter test plan. Readers are required to read through this chapter carefully and understand various configuration points of each components. We cover configurations and use of test plans, thread groups, pre-processors, logic controllers, timers, samplers, assertions, post-processors, properties, variables, and user defined variables.

By the end of this chapter, you will have a good idea of the various components of a typical test plan and will have a good understanding of the configurations and uses of these components. You will also be able to develop a standard JMeter test script. Those who are already familiar with any of these components can skip the appropriate sections and move forward to the next section.

Test Plan

The *test plan* is the root element of the JMeter test. It is a container that holds JMeter components like the *thread group*, *logic controller*, *sampler*, *listener*, *timer*, *assertion*, and *config element*. A test plan (see Figure 5-1) consists of one or more thread groups.

Test Plan

Name:	Test Plan
Comments:	
User Defined Variables	
Name:	Value
<input type="button" value="Detail"/> <input type="button" value="Add"/> <input type="button" value="Add from Clipboard"/> <input type="button" value="Delete"/> <input type="button" value="Up"/> <input type="button" value="Down"/>	
<input type="checkbox"/> Run Thread Groups consecutively (i.e. run groups one at a time) <input type="checkbox"/> Run tearDown Thread Groups after shutdown of main threads <input type="checkbox"/> Functional Test Mode (i.e. save Response Data and Sampler Data)	
Selecting Functional Test Mode may adversely affect performance.	
Add directory or jar to classpath	<input type="button" value="Browse..."/> <input type="button" value="Delete"/> <input type="button" value="Clear"/>
Library	

Figure 5-1. Test plan configuration

Configuration

Figure 5-1 shows the configuration of a test plan. The configuration points are described next.

- **User Defined Variables:** These are defined by using *name/value* pair(s). Click the Add button to create a new name/value pair. If you have more than one, create a list of name/value pairs, one per line, in a text file. Select all of them and then click on Add Clipboard to add all of them at the same time. Select one or more and click on the Delete button to delete. Use the Up and Down buttons to change the order.
- **Run Thread Groups Consecutively (i.e. Run Groups One at a Time):** A test plan can have more than one thread group. If this checkbox is enabled, then the thread groups are executed one after the other. If this checkbox is not enabled (the default), the thread groups are executed in parallel.
- **Run tearDown Thread Groups After Shutdown of Main Threads:** If this checkbox is selected, then tearDown thread groups executes after the test has finished executing its regular thread group. It is used for doing things post test execution for reporting purposes or performing cleanup operations.

- **Functional Test Mode (i.e. Save Response Data and Sampler Data):** If this checkbox is selected, then sampler requests and response data are saved in the listeners. This allows you to verify that the test is working as expected. The use of this feature is not recommended.
- **Add Directory or JAR to Classpath:** You can add a folder or a JAR file to the classpath and JMeter can load these classes. For consistent behavior, restart JMeter after modifying this.

We explore serial execution of thread groups, parallel execution of thread groups, and user defined variables in the following sections.

Serial Execution of Thread Groups

If the Run Thread Groups Consecutively (i.e. Run Groups One at a Time) checkbox is enabled, then for test plans that have multiple thread groups, the next thread group will start executing only after the prior thread group has finished execution.

Let's illustrate this by the following example.

Follow these steps or download [SerialExecutionOfThreadGroupTestPlan.jmx](#).¹

1. Create a test plan and give it a meaningful name, such as `Serial Execution of Thread Group Test`. Enable the **Run Thread Groups Consecutively** checkbox (see Figure 5-2).

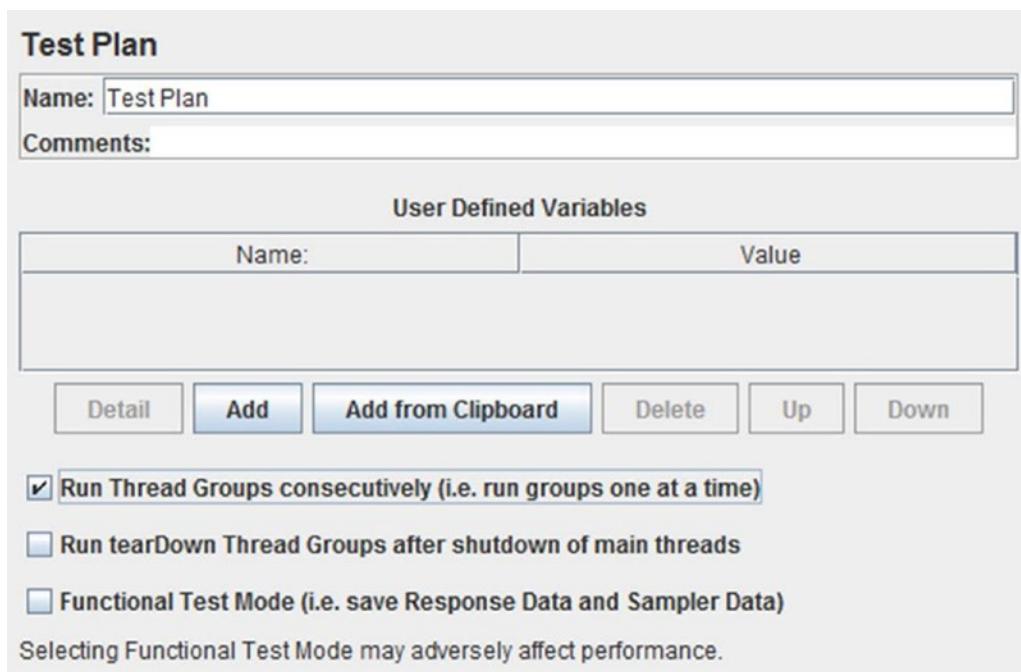


Figure 5-2. Serial execution of thread groups

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/test-plan/SerialExecutionOfThreadGroupTestPlan.jmx

2. Click on Test Plan and go to Edit ► Add ► Threads (Users). Add Thread Group. Configure the **Name** as Thread Group A and **Loop Count** as 4 (see Figure 5-3).

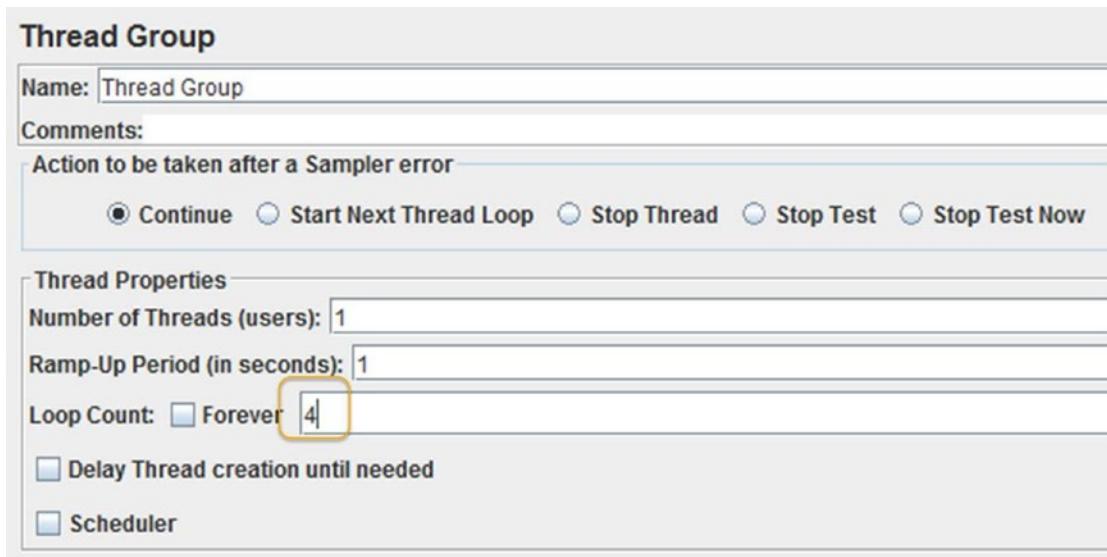


Figure 5-3. Thread group loop count

3. Click on Thread Group A and go to Edit ► Add ► Sampler. Add HTTP Request. Configure **Name** as Alpha, **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /jmeter/alpha.
4. Click on Test Plan and go to Edit ► Add ► Threads (Users). Add Thread Group. Configure the **Name** as Thread Group B and **Loop Count** as 4.
5. Click on Simple Controller and go to Edit ► Add ► Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
6. Click on Test Plan and go to Edit ► Add ► Listener. Add View Results Tree.
7. Save the test plan.
8. Run the test.

The results are as shown in Figure 5-4.

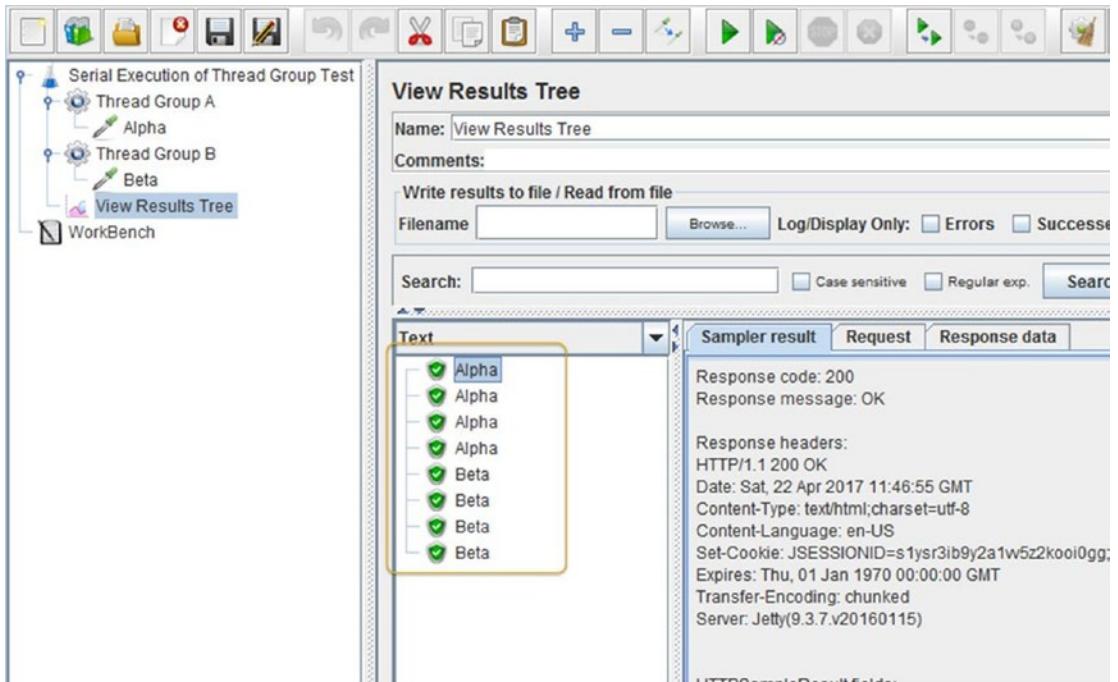


Figure 5-4. Serial execution of threads results

Parallel Execution of Thread Groups

If the Run Thread Groups Consecutively (i.e. Run Groups One at a Time) checkbox is not enabled, then for test plans that have multiple thread groups, all the thread groups will start executing at the same time.

Let's illustrate this in the following example.

Follow these steps or download [ParallelExecutionOfThreadGroupTestPlan.jmx](#).²

1. Open `SerialExecutionOfThreadGroupTestPlan.jmx`.

²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/test-plan/ParallelExecutionOfThreadGroupTestPlan.jmx

2. Click on the test plan and uncheck the **Run Thread Groups Consecutively** checkbox (see Figure 5-5).

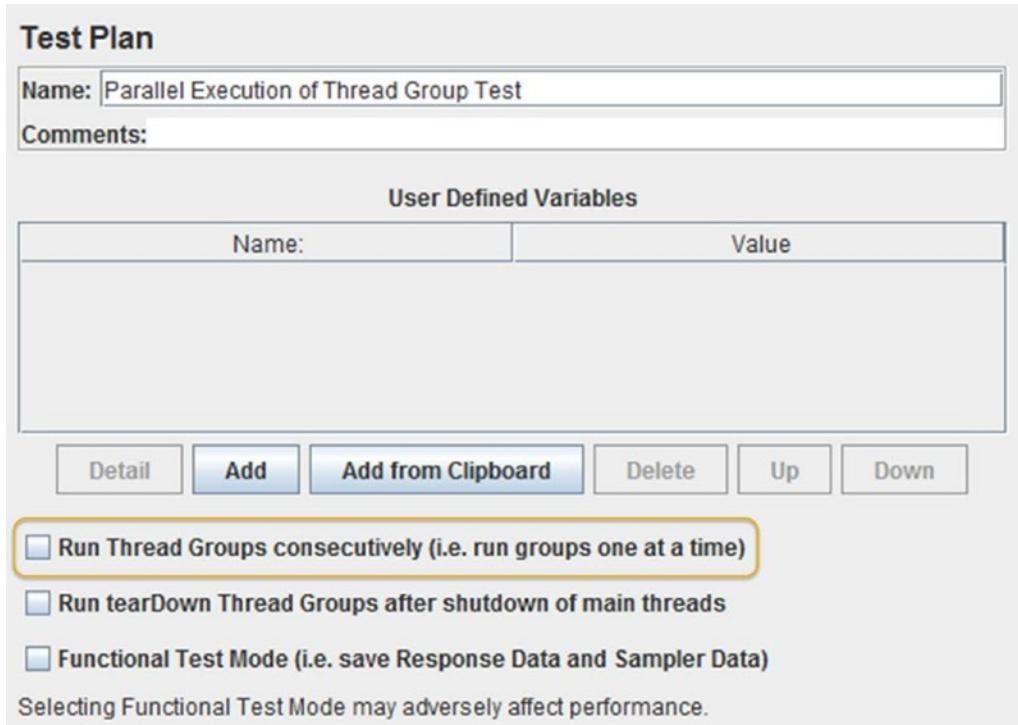


Figure 5-5. Parallel execution of thread groups

3. Run the test (see Figure 5-6).

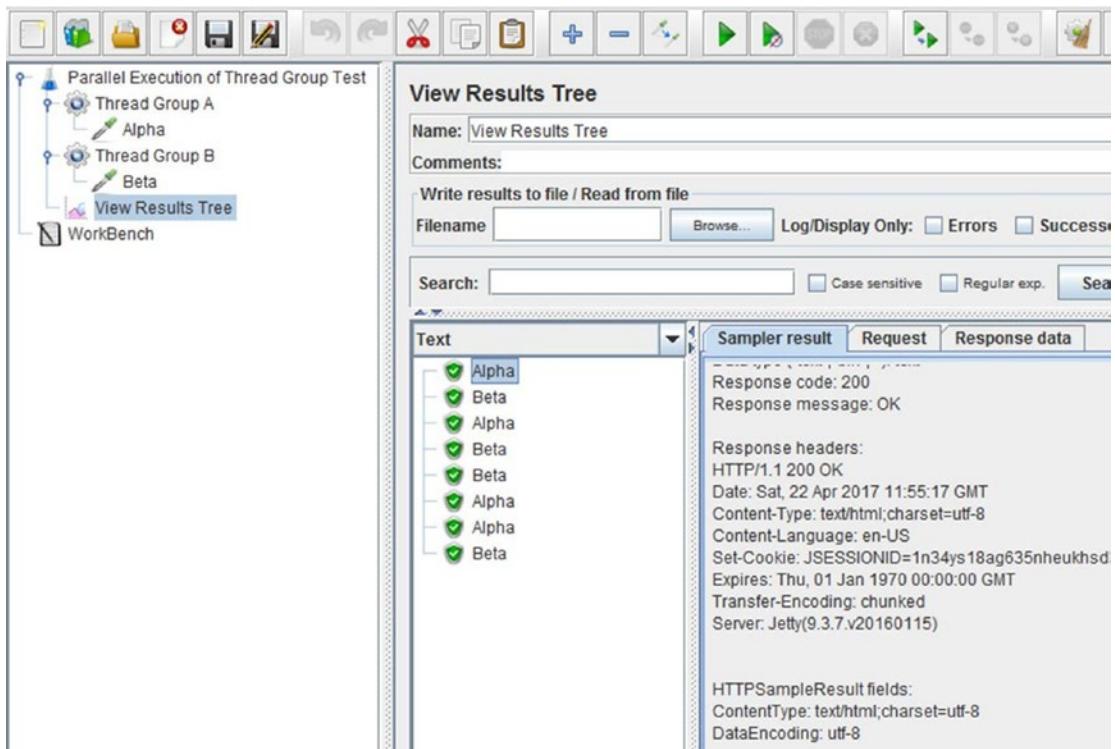


Figure 5-6. Parallel execution of threads results

The results are as shown in Figure 5-6.

Tip Pay attention to the Run Thread Groups Consecutively (i.e. Run Groups One at a Time) checkbox.

User Defined Variables

User defined variables are associated with values that are constant throughout the execution of the test. For example, if the server name of the HTTP Request is defined as a user defined variable, later when the server name changes, you just need to change the value of the Server Name field. The important thing to note is that the value of the server name does not change throughout the execution of the test script.

User defined variables have a scope across the entire test plan. These are copied to the variables in each of the thread groups before the test starts executing.

Whether the thread groups are being executed consecutively or in parallel, these user defined variables are available and can be utilized.

Note Last section of this chapter “Properties and Variables” describes UDV with an example.

Thread Group

The Thread Group element is the starting point of execution. All elements can be the child elements of a test plan or a thread group (see Figure 5-7), except for controllers and samplers, which can only be the child elements of a thread group.

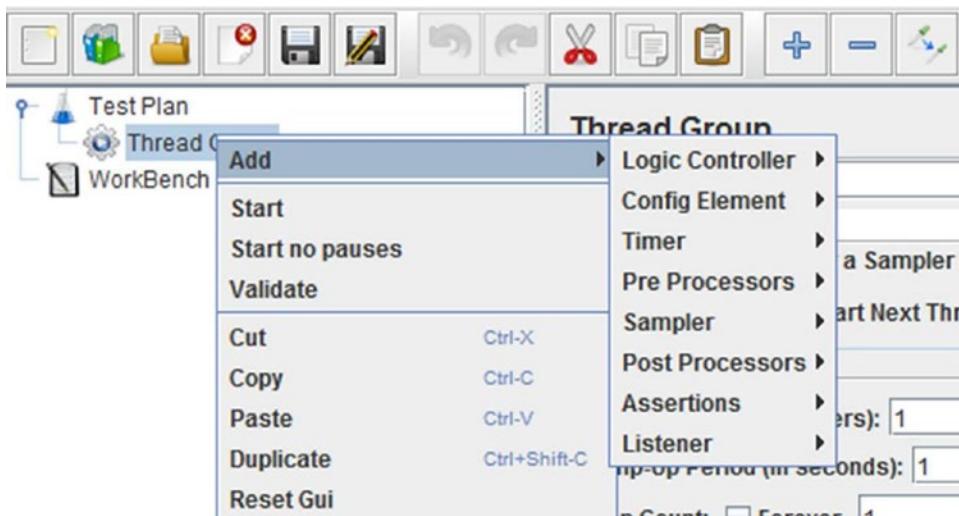


Figure 5-7. Thread group contents

The thread group is simulating the load generated by users performing a use-case. A test plan can have multiple thread groups, thus simulating multiple use-cases.

Although uncommon, data can be exchanged between thread groups.

The Number of Threads (Users) option is used to specify the number of users accessing the web application. When the test starts initially, threads are spawned based on the configuration option called Ramp-Up Period (in Seconds).

You can also schedule tests to run at a particular time.

When a test fails, the behavior to continue or to stop is configurable (see Figure 5-8).

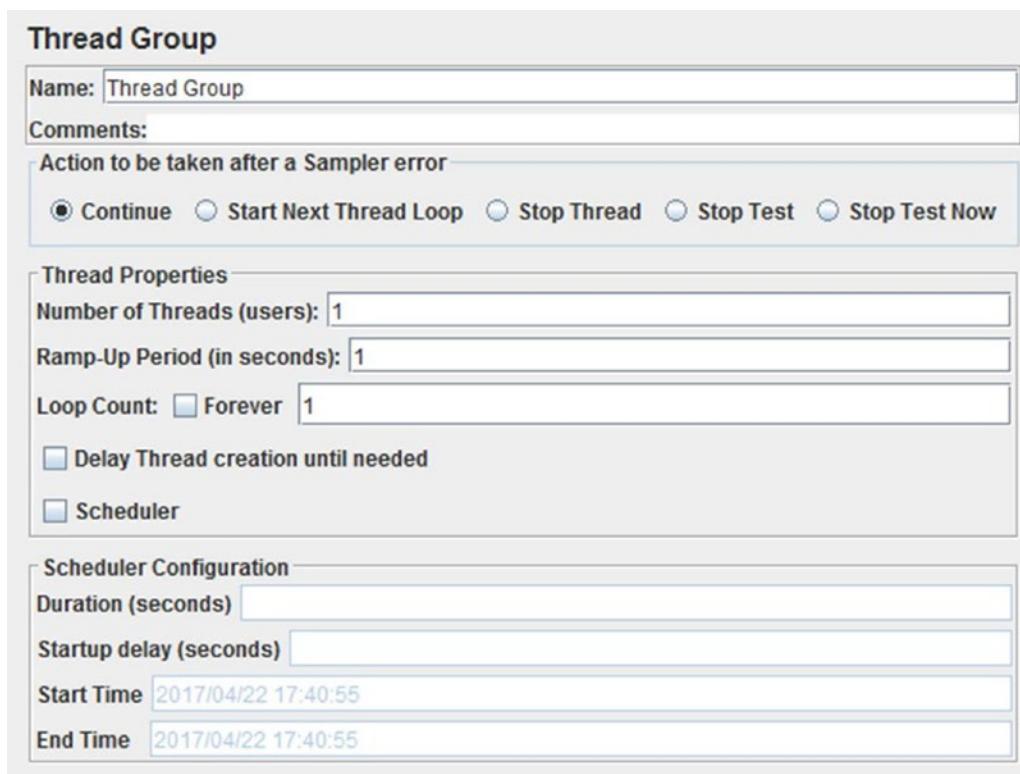


Figure 5-8. Thread group

Let's explore each of these options in the following section.

Thread Properties

The Thread Properties option (see Figure 5-9) gives you the flexibility to simulate a realistic load.

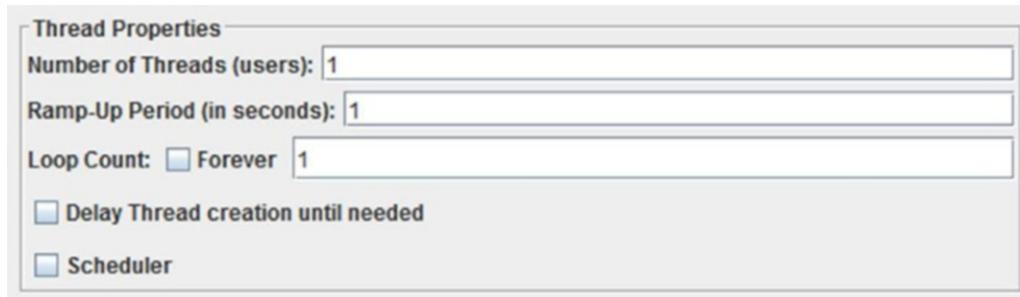


Figure 5-9. Thread group

- **Number of Threads (Users):** This is the number of users we want to use for load testing a web application.
- **Ramp-Up Period (in Seconds):** This is the time after which all threads will be active.

Assume that you want to start 10 threads, with a thread starting every second, then configure the thread group with **Number of Threads (users)** as 10 and **RampUp Period (in Seconds)** as 10. With this configuration, 10 threads will start in 10 seconds and all threads will be active after 10 seconds.

Let's see how to use Number of Threads (Users) and Ramp-Up Period (in Seconds) using the following example.

Follow these steps or download ThreadGroupTestPlan.jmx.

1. Create a test plan and give it a meaningful name, such as Thread Group Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (users)** as 10 and **Ramp-Up Period (in seconds)** as 10 (see Figure 5-10).

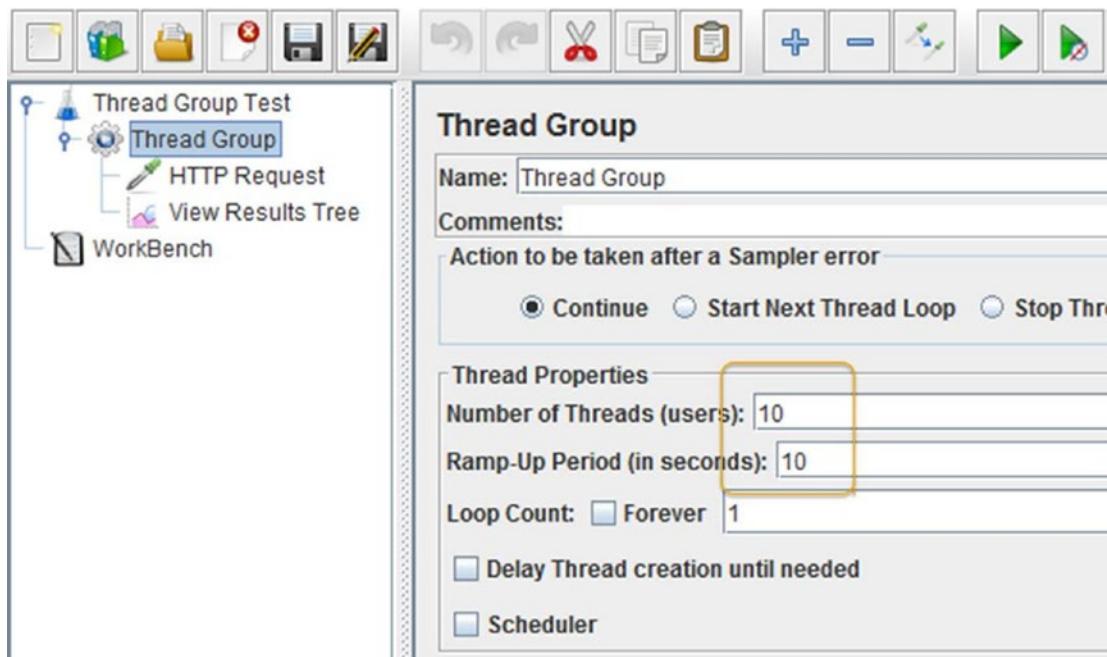


Figure 5-10. Thread group test

3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
5. Save the test plan.
6. Run the test.

The results will be similar to those shown in Figures 5-11 and 5-12.

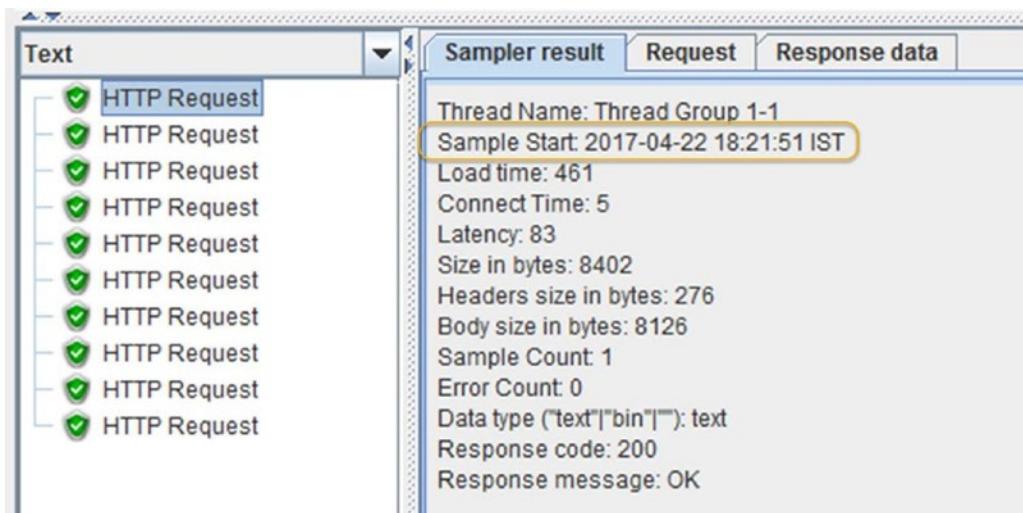


Figure 5-11. Thread group test results - thread one

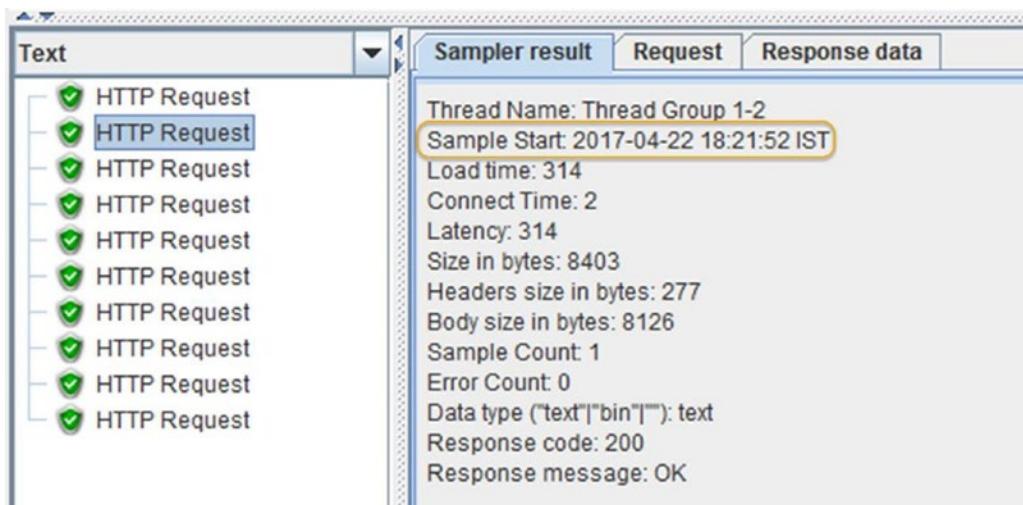


Figure 5-12. Thread group test results - thread two

7. Click on each of the responses and look for Sample Start. This will show you when the thread started. You will find that each thread was started after a gap of 1 second.
- **Loop Count:** This option controls the number of iterations that the thread group runs.

Let's see how to use Loop Count using the following example.

Follow these steps or download LoopedThreadGroupTestPlan.jmx.³

1. Create a test plan and give it a meaningful name, such as Looped Thread Group Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (users)** as 1, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 2 (see Figure 5-13).

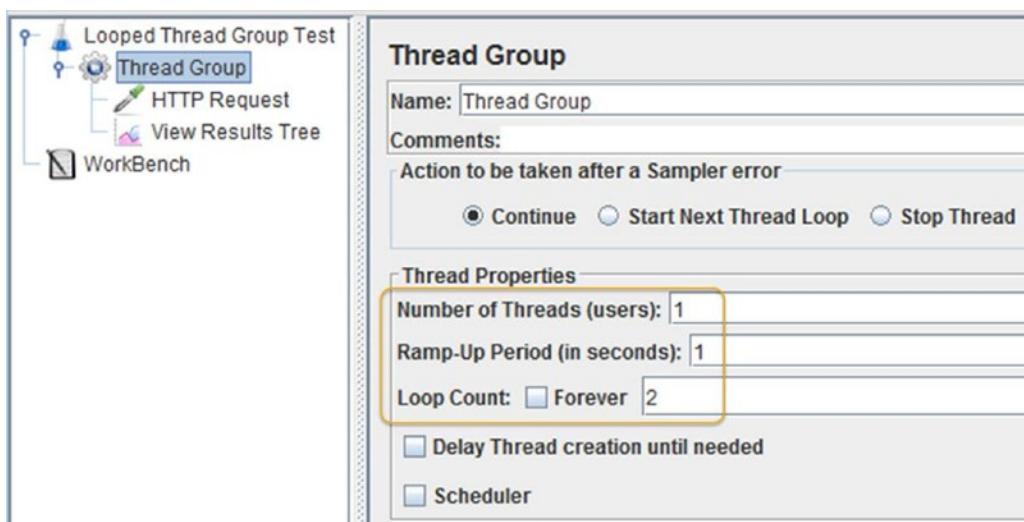


Figure 5-13. Looped thread group test

3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
5. Save the test plan.

³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/LoopedThreadGroupTestPlan.jmx

6. Run the test. The results are shown in Figure 5-14.

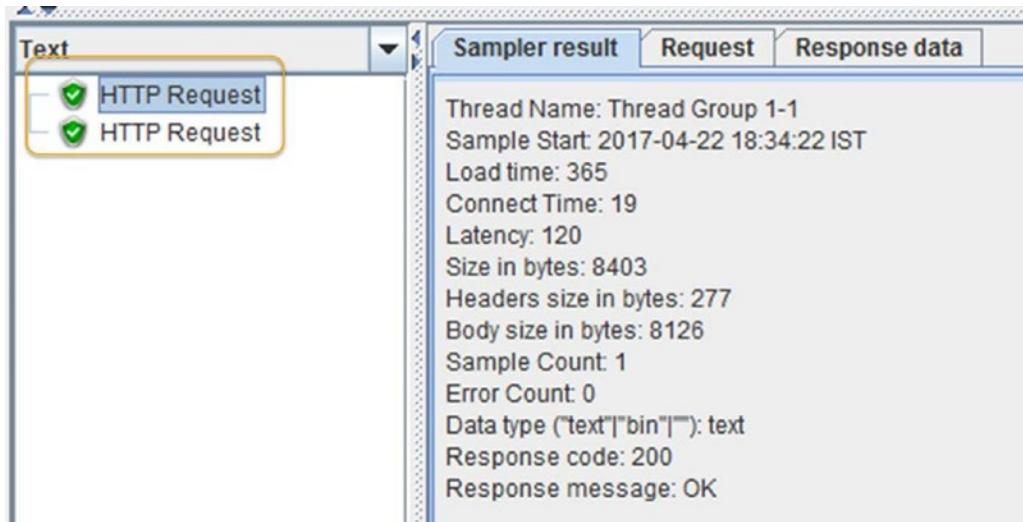


Figure 5-14. Looped thread group test results

7. Since we have configured **Loop Count** as 2, two responses are captured inside *View Results Tree*.
 - **Forever:** By selecting this checkbox, we are telling JMeter that the execution should iterate until the user clicks the Stop or Shutdown buttons.

Let's see how to use Forever option using the following example.

Follow these steps or download `ForeverRunningThreadGroupTestPlan.jmx`.⁴

1. Create a test plan and give it a meaningful name, such as **Forever Running Thread Group Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group. Configure **Number of Threads (users)** as 1, **Ramp-Up Period (in seconds)** as 1 and click the **Forever** checkbox (see Figure 5-15).

⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ForeverRunningThreadGroupTestPlan.jmx

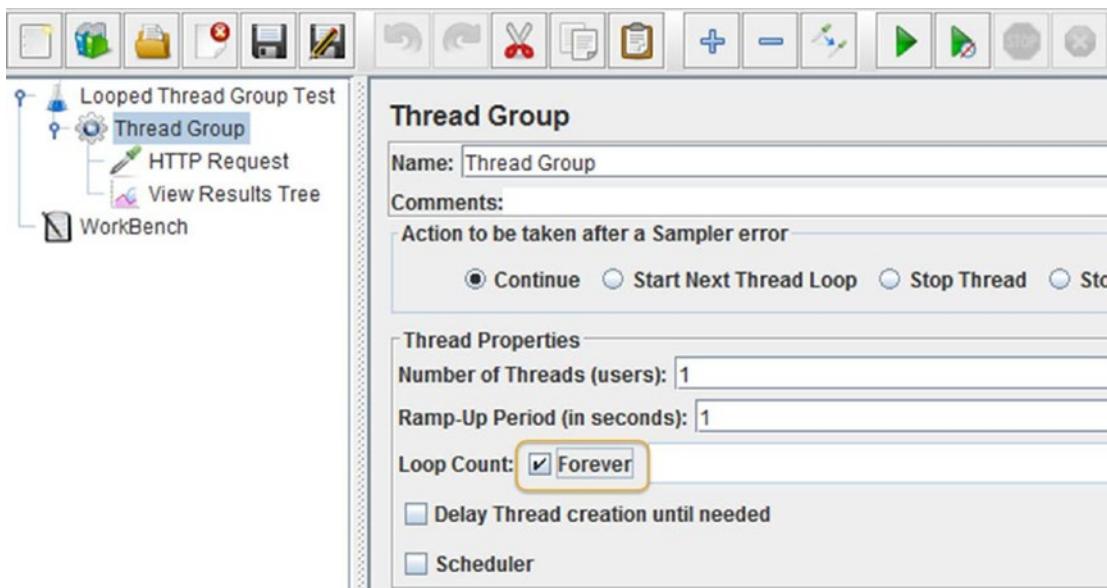


Figure 5-15. Forever thread group test

3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
5. Save the test plan.
6. Run the test.
7. The test will never end, with responses being added to the View Results Tree Listener.
8. Click on the Stop button to stop the test immediately or click on the Shutdown button to terminate the test gracefully.

Scheduler

The other properties of the thread group are Scheduler and Delayed Thread Creation until Needed (see Figure 5-16).

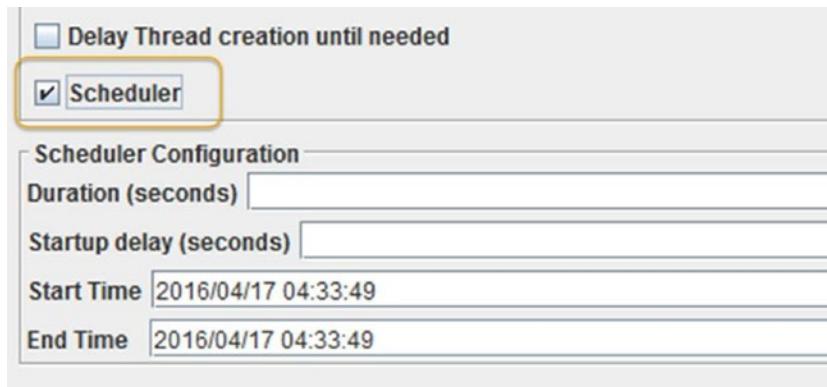


Figure 5-16. Scheduler

Web site traffic load patterns vary based on the time of day. This can be simulated by configuring the scheduler to run the test at a particular time.

Let's see this in the following example.

Follow these steps or download `ScheduledThreadGroupTestPlan.jmx`.⁵

1. Create a test plan and give it a meaningful name, such as **Scheduled Thread Group Test**.
2. Click on Test Plan and go to **Edit ▶ Add ▶ Threads (Users)**. Add Thread Group. Configure **Number of Threads (users)** as 1, **Ramp-Up Period (in seconds)** as 1, and then click the **Forever** checkbox (see Figure 5-17).
3. Click on the Scheduler checkbox. Let's say the system time is 07:18:10 and you want the test to run 5 minutes from now. Configure the **Start Time** to 07:23:10 and **End Time** to 07:23:11. Here, you are telling JMeter to run the test 5 minutes from now and to run it for .01 seconds (see Figure 5-17).

⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ScheduledThreadGroupTestPlan.jmx

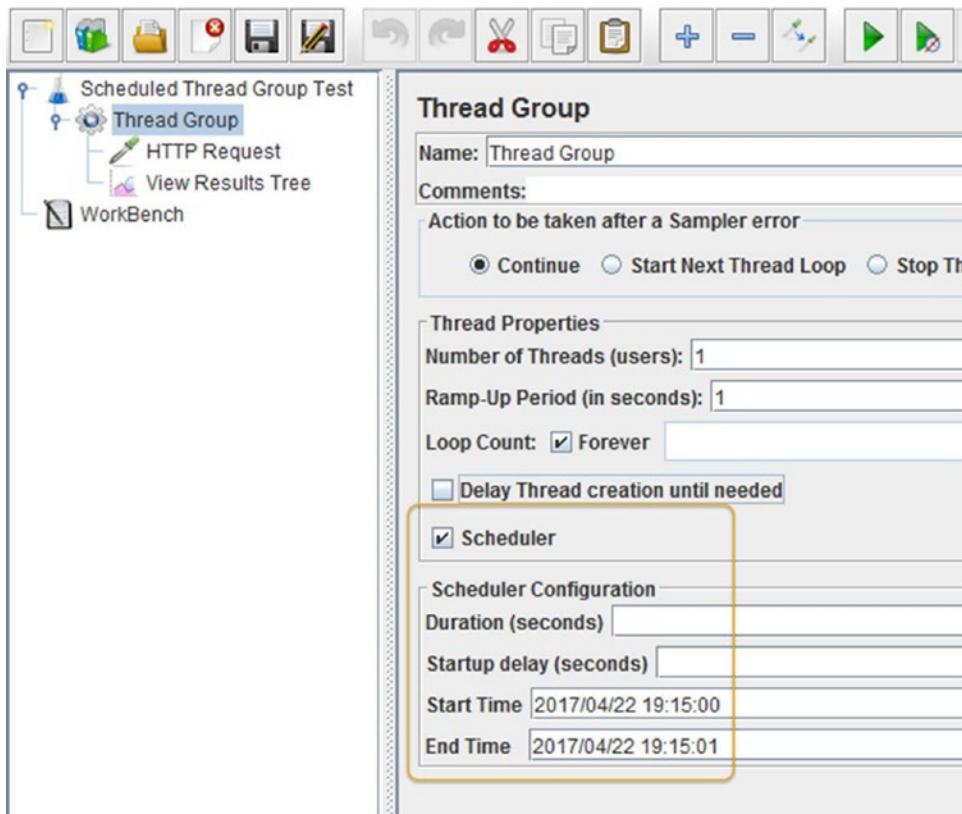


Figure 5-17. Scheduled thread group test

4. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
5. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
6. Save the test plan.
7. Run the test. The test will not run immediately, as execution was scheduled to start after 5 minutes. Wait for 5 minutes and you will see responses under the View Results Tree. Look at the Sampler Start time; it will show the configured value (see Figure 5-18).

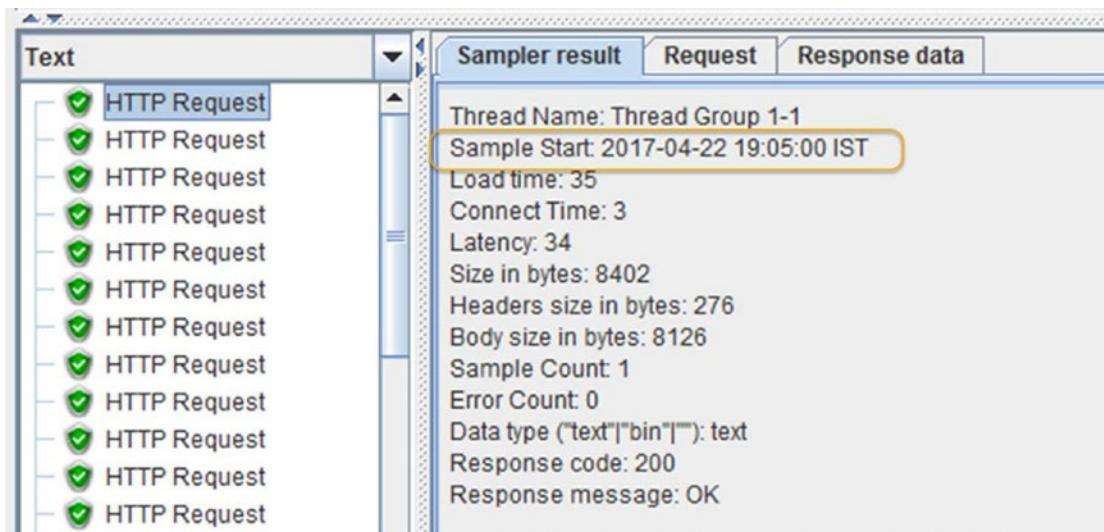


Figure 5-18. Scheduled thread group test results

The test will stop automatically after .01 second.

The Sampler Start Time is ignored if Startup Delay (Seconds) is specified. Let's see this using the following example.

1. Open ScheduledThreadGroupTestPlan.jmx.
2. Click on Thread Group and configure **Duration (Seconds)** as 2 and **Startup delay (seconds)** as 10 (see Figure 5-19).

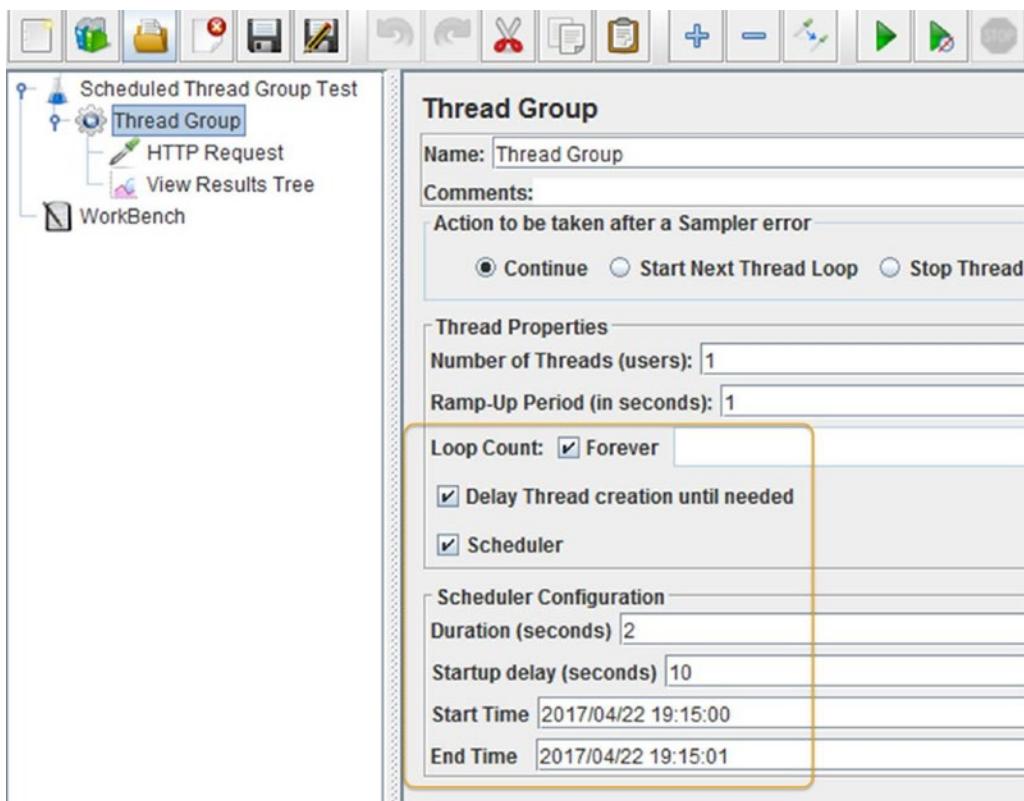


Figure 5-19. Startup delay thread group test

3. Save the test plan.
4. Click on the Start button. The test will not start immediately, but will start automatically after 10 seconds and end after 2 seconds (see Figure 5-20).

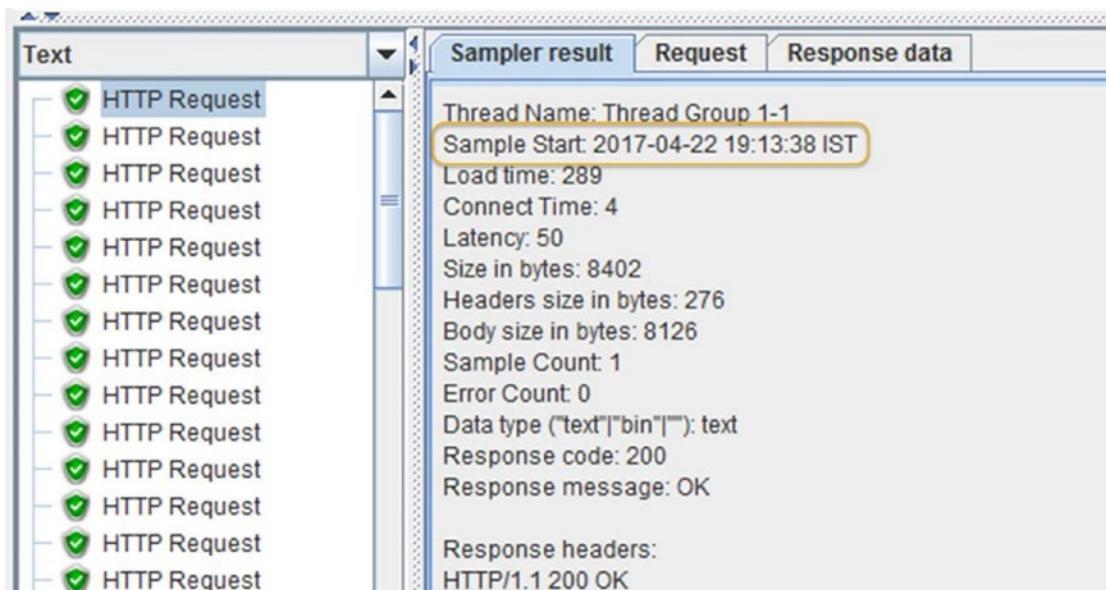


Figure 5-20. Startup delay thread group test results

Note The configuration option called Delayed Thread Creation Until Needed is used to minimize the number of threads. This is an optimization feature.

Action After Sampler Error

When a sampler fails, the thread group execution behavior is configurable.

- Continue: If selected, the execution will continue irrespective of the error.

Let's see this action in the following example.

Follow these steps or download `ThreadGroupActionsTestPlan.jmx`.⁶

1. Create a test plan and give it a meaningful name, such as `Thread Group Actions Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group. Configure **Number of Threads (users)** as 1, **Ramp-Up Period (in seconds)** as 1, then click the **Forever** checkbox and select **Actions to be taken after Sampler error** as Continue. Give it the name `Thread Group A`.
3. Click on thread group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as `/dt`. Give it the name `HTTP Request-A#1`.

⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ThreadGroupActionsTestPlan.jmx

4. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (users)** as 1, **Ramp-Up Period (in seconds)** as 1, select **Forever** checkbox and select **Actions to Be Taken After Sampler Error** as Continue. Give it the name Thread Group B.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt. Give it the name HTTP Request-B#1.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /wrong-dt. Give it the name HTTP Request-B#2.
7. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt. Give it the name HTTP Request-B#3.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 5-21).

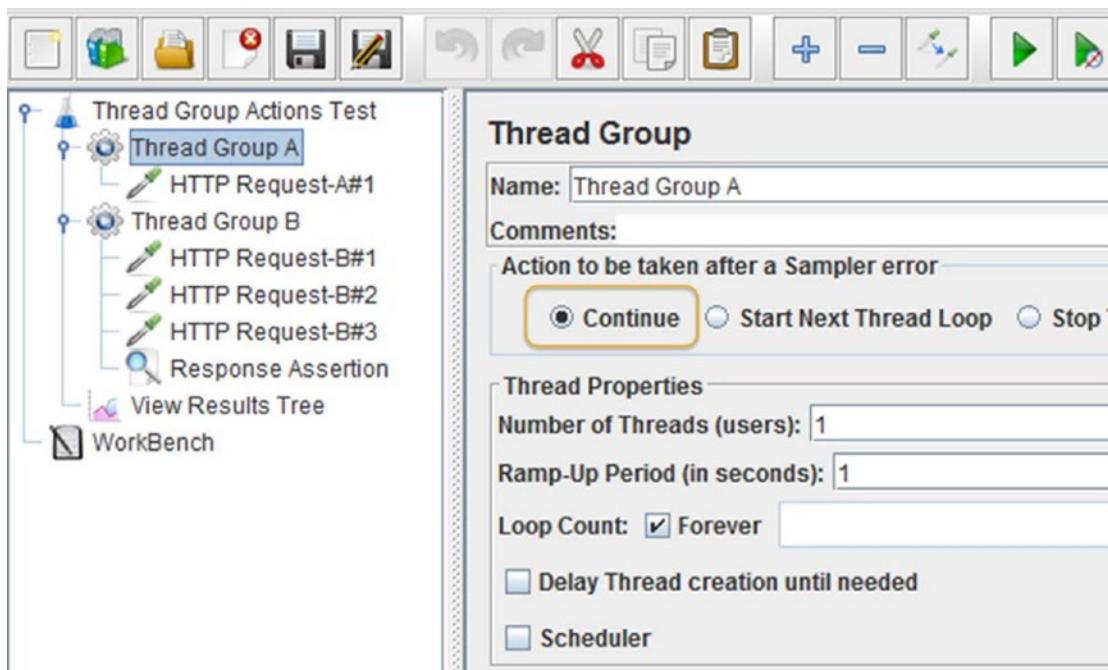


Figure 5-21. Continue thread group test

9. Save the test plan.
10. Run the test.
11. Click on View Results Tree. You will notice that there was an error in the sampler, as indicated by the red color. Despite the error, the test continues to execute (see Figure 5-22).

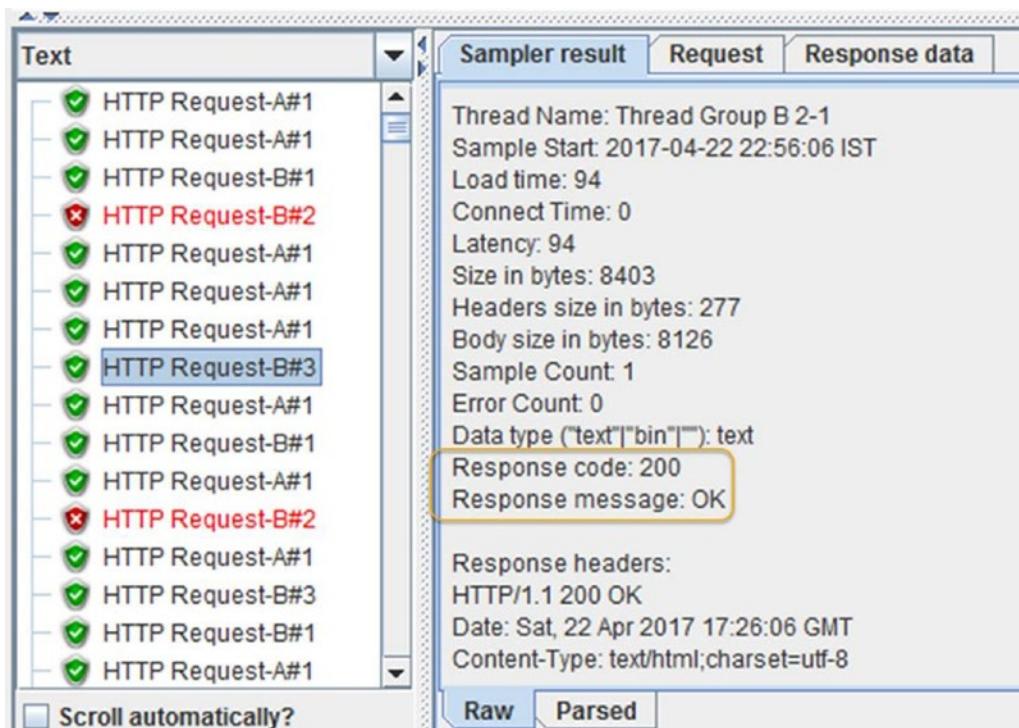


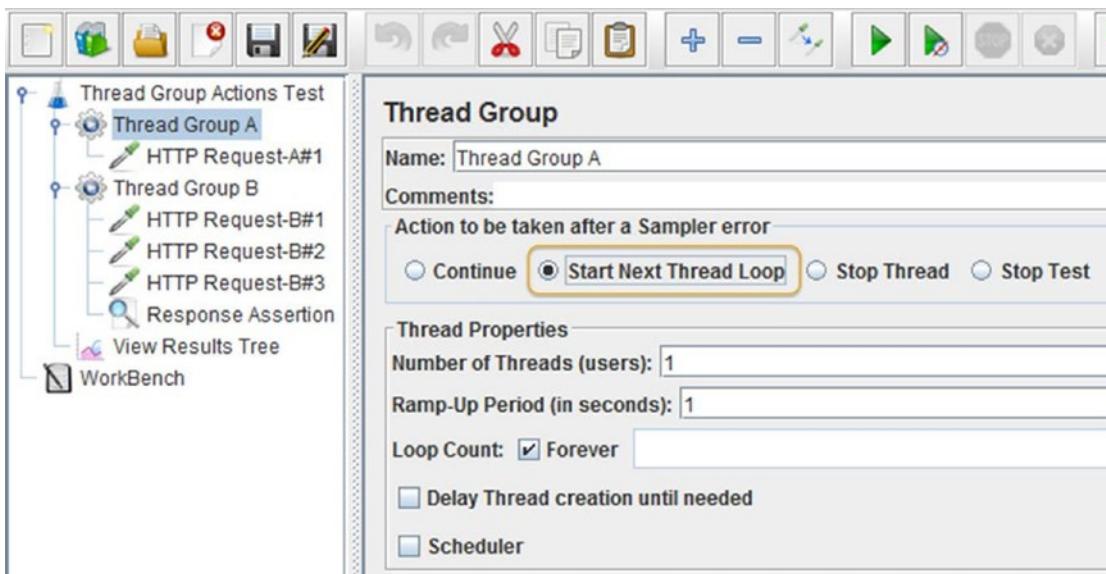
Figure 5-22. Continue thread group test results

- **Start Next Thread Loop:** If selected, when a sampler error occurs, the current thread loop is terminated and the next thread loop is started.

Follow these steps to observe this behavior.

1. Open `ThreadGroupActionsTestPlan.jmx`.⁷
2. Click on Thread Group B. Configure Action to Be Taken After a Sampler Error as Start Next Thread Loop (see Figure 5-23).

⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ThreadGroupActionsTestPlan.jmx

**Figure 5-23.** Start next thread loop test

3. Save the test plan.
4. Run the test.
5. Click on View Results Tree. You will see that there was an error at HTTP Request-B#2, after which the execution of the thread stops, never executing HTTP Request-B#3. The next iteration of the thread group starts (see Figure 5-24).

The screenshot shows the 'Text' view of the 'View Results Tree' component. It lists several 'HTTP Request' entries, some successful (green checkmark) and some failed (red X). The failed entries are 'HTTP Request-B#2' (highlighted with a blue border) and 'HTTP Request-B#3'. To the right, the 'Sampler result' tab of the 'View Results Tree' panel is active, displaying detailed results for the failed 'HTTP Request-B#2'. It shows an 'Error Count: 1', 'Data type ("text"|"bin"|"'): text', 'Response code: 404', and 'Response message: Not Found'. Below this, 'Response headers:' are listed: 'HTTP/1.1 404 Not Found', 'Date: Sat, 22 Apr 2017 17:40:24 GMT', 'Content-Type: text/html;charset=iso-8859-1', 'Cache-Control: must-revalidate,no-cache,no-store', 'Content-Length: 324', and 'Server: Jetty(9.3.7.v20160115)'. At the bottom, 'HTTPSampleResult fields:' are listed: 'ContentType: text/html;charset=iso-8859-1' and 'DataEncoding: iso-8859-1'.

Figure 5-24. Start next thread group test results

6. Click on the Stop button to manually stop the test.
- **Stop Thread:** If selected, when a sampler error occurs, the thread is terminated.

Any other threads in the thread group will continue execution.

Follow these steps to observe this behavior.

1. Open ThreadGroupActionsTestPlan.jmx.⁸
2. Click on Thread Group B. Configure **Action to be Taken After a Sampler Error** as Stop Thread (see Figure 5-25).

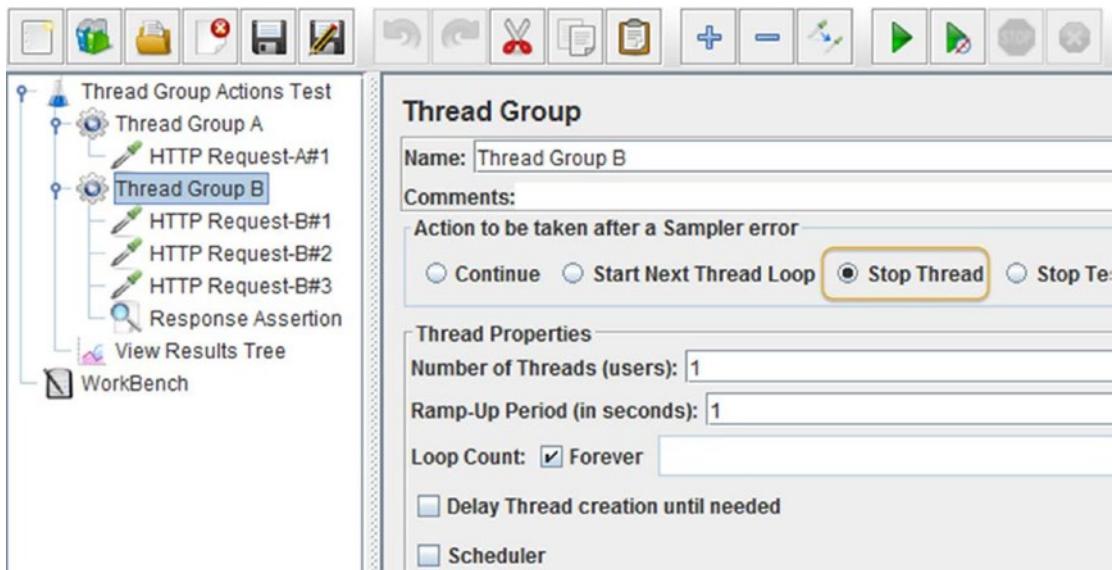


Figure 5-25. Stop thread test

3. Save the test plan.
4. Run the test. Click on View Results Tree. You will see that, after the sampler error, the test will never execute samplers from Thread Group B (see Figure 5-26).

⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ThreadGroupActionsTestPlan.jmx

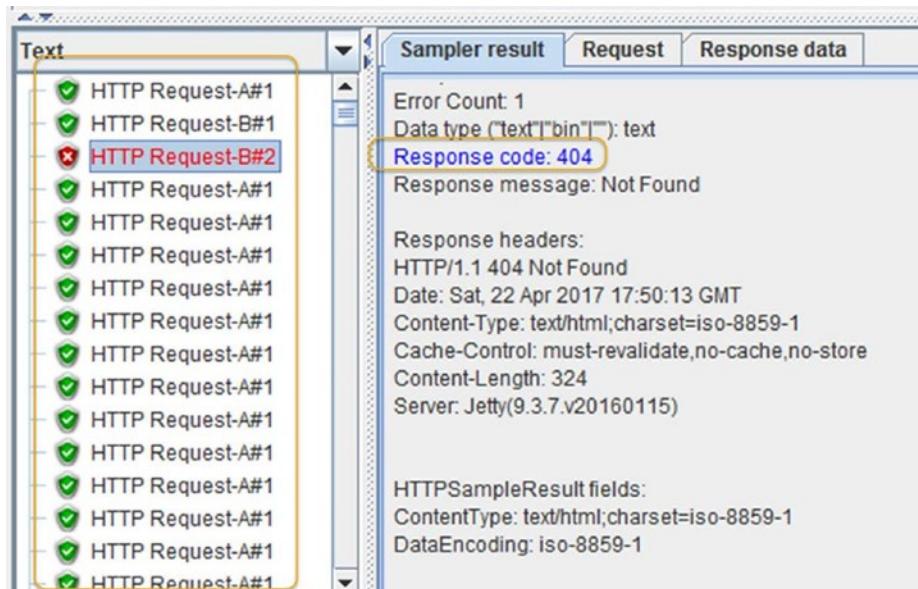


Figure 5-26. Stop thread test results

- **Stop Test:** If selected, this will stop the test entirely but allow the currently executing samplers to finish.

Follow these steps to observe this behavior.

1. Open ThreadGroupActionsTestPlan.jmx⁹.
2. Click on Thread Group B. Configure **Action to be taken after a Sampler error** as **Stop Test** (see Figure 5-27).

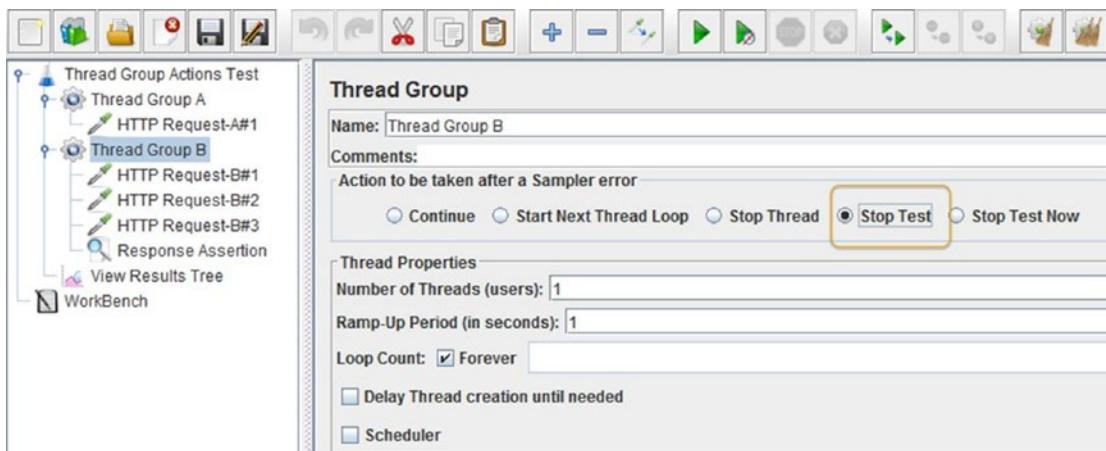


Figure 5-27. Stop test

⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ThreadGroupActionsTestPlan.jmx

3. Save the test plan.
4. Run the test. Click on View Results Tree. You will see that, after the sampler error at HTTP Request-B#2, the test stops just after allowing the currently executing sampler HTTP Request-A#1 to finish (see Figure 5-28).

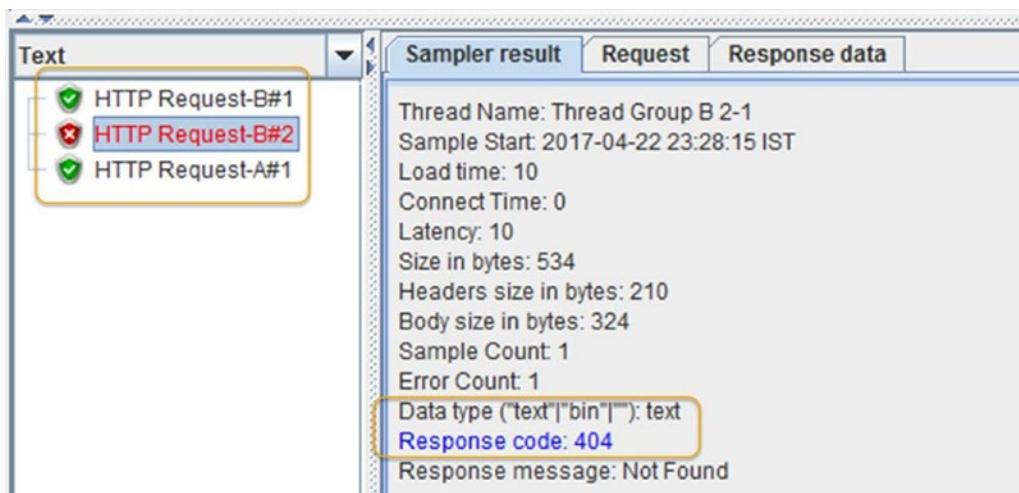


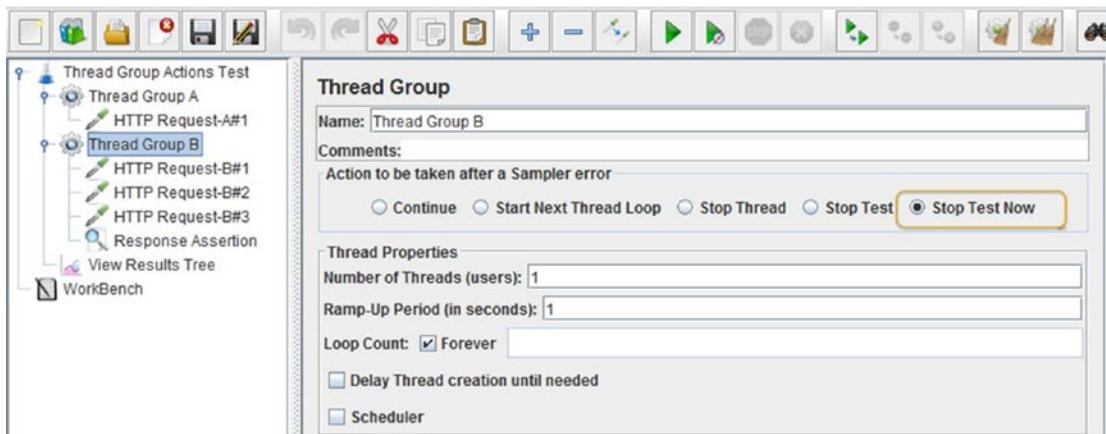
Figure 5-28. HTTP request sampler result

- **Stop Test Now:** If selected, this will stop the test abruptly without waiting for the currently executing samplers to finish.

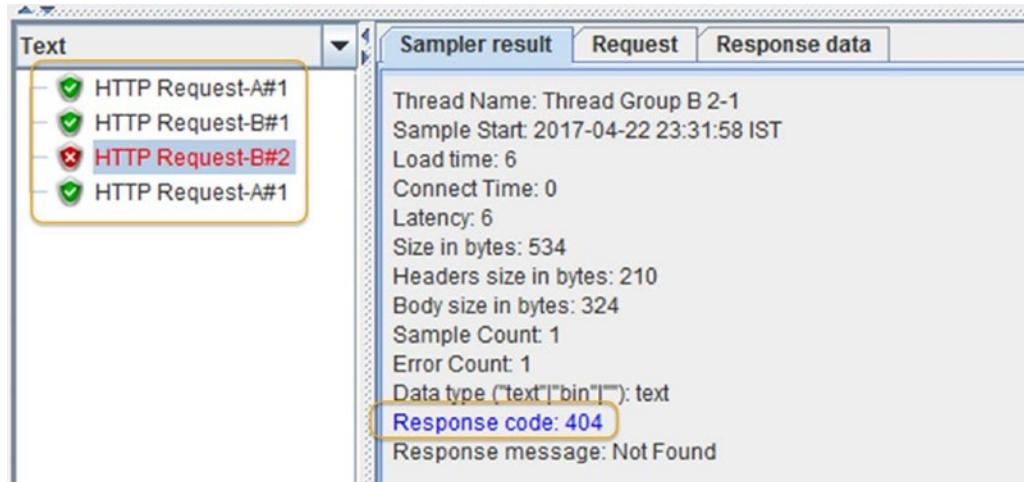
Follow these steps to observe this behavior.

1. Open ThreadGroupActionsTestPlan.jmx¹⁰.
2. Click on Thread Group B. Configure **Action to be Taken After a Sampler Error** as Stop Test Now (see Figure 5-29).

¹⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/thread-group/ThreadGroupActionsTestPlan.jmx

**Figure 5-29.** Stop test now test

3. Save the test plan.
4. Run the test.
5. Click on View Results Tree. You will see that, after the sampler error at HTTP Request-B#2, the test stops abruptly without allowing the currently executing samplers HTTP Request-A#1 to finish (see Figure 5-30).

**Figure 5-30.** Stop test now results

Pre-Processors

Pre-processors take the request and modify (substitution, enhancement, dereferencing variables etc.) it before the sampler sends it to the server.

Certain web browsers do not support cookies. In such cases, the `session-id` is passed as a parameter in the URL. The *HTTP URL Re-writing Modifier* pre-processes the sampler requests and appends the necessary `session-id` information to every embedded URL before the request is sent to the server.

Pre-processors are executed after the Config Elements and before the timers and samplers as per JMeter test plan execution order.

HTTP URL Re-Writing Modifier

The *HTTP URL Re-Writing Modifier* (see Figure 5-31) is used to store session ID. It is added as a child to the thread group. It will automatically determine the session ID of the web application being tested based on the session variable name used in the web application.

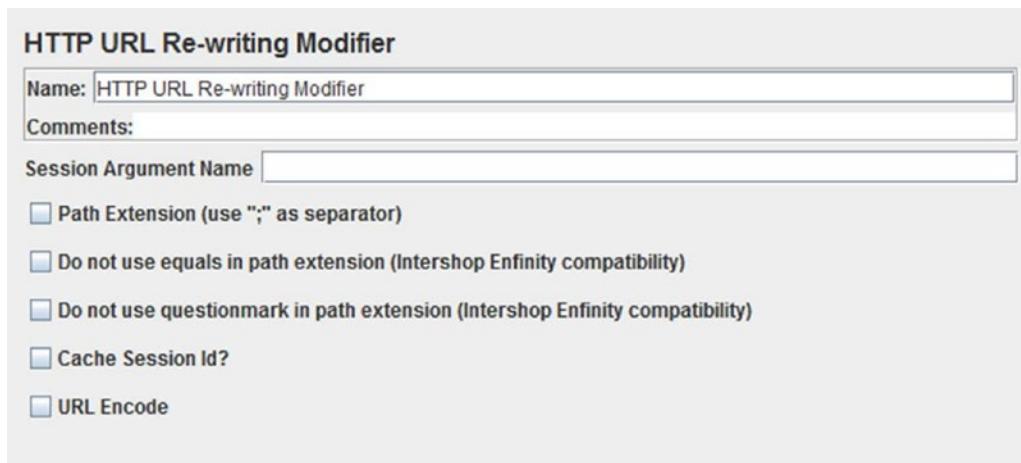


Figure 5-31. HTTP URL re-writing modifier pre-processor

- **Session Argument Name:** Variable used to store the session ID in the web application.
- **Path Extension (Use ";" as Separator):** Should be selected only when the URL parameters are separated by semicolons (;).
- **Do not use equals in path extension:** Should be selected if you do not want name/value pair separated by equals signs (=).
- **Do not use question mark in path extension:** Should be selected if you do not want name/value pair separated by question marks (?).
- **Cache Session Id?:** Should be selected if you want to use the Caching feature of JMeter.
- **URL Encode:** Should be selected if your URL is using any encoding Let's illustrate this with the following example.

Before starting the test, let's set up the browser for recording steps on JMeter.

Follow these steps or download PreProcessorHTTPURLReWritingTestPlan.jmx.¹¹

1. Create a test plan and give it a meaningful name, such as Pre Processor HTTP URL Re Writing Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Elements. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /user/signIn and **Method** as POST. Under the parameters, set **Send Parameters With the Request** to **Name:Value** pairs, email to user100@dt.com, and password to user100.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure the path to /order/addToCart and the method to POST. Under the parameters, **Send Parameters With the Request**: add **Name:Value** pair, product as 1.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure the path to /user/addAddress and the method to POST. Under the parameters, set **Send Parameters With the Request** to **Name:Value** pairs, set checkOutFlow to true, and leave addressId empty. Then set line1 as AddressLine1, line2 as AddressLine2, city as Foster, state as CA, and zip as 12345.
7. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure the path to /user/addCard and the method to POST. Under the parameters, set **Send Parameters With the Request**: to **Name:Value** pair, checkOutFlow to true, and leave cardId empty. Then set nameOnCard to user100, cardNumber to 1234123412341234, and expirationMonthYear to 12/2016.
8. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure the path to /order/placeOrder and the method to GET.
9. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure the path to /user/signOut and the method to GET.
10. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
11. Save the test plan.
12. Run the test.

Login to the Digital Toys Inc. web application using user100@dt.com/user100 and find out if the order is placed successfully. You will find out that there was no order placed for this user. This is because the user session information was not passed to the server.

Follow these steps to add the *HTTP URL Re-writing Modifier*.

1. Click on Thread Group and go to Edit ▶ Add ▶ Pre Processors. Add HTTP URL Re-writing Modifier. Configure the session argument name to jsessionid, and then click the checkbox for **Path Extension (use ";" as Separator)** (see Figure 5-32). Leave all other checkboxes unchecked.

¹¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/pre-processors/PreProcessorHTTPURLReWritingTestPlan.jmx

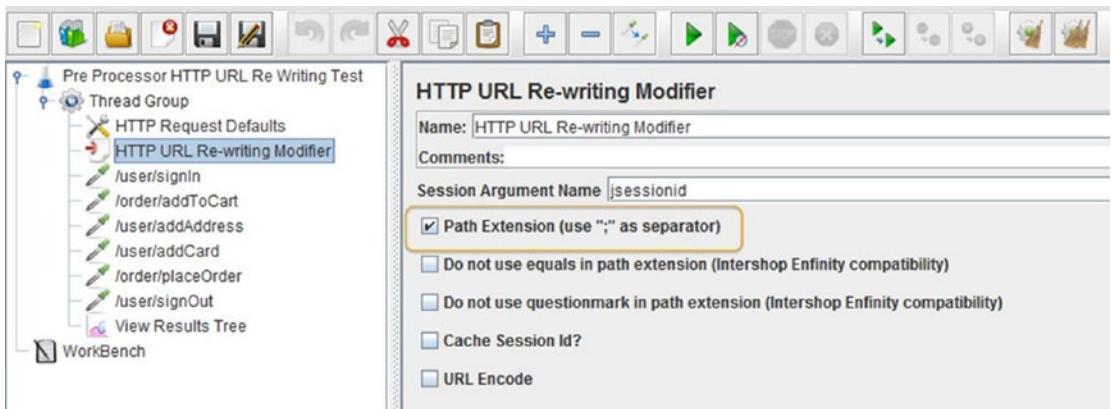


Figure 5-32. HTTP URL re-writing modifier

2. Run the test.
3. Verify the results of the test script execution by logging into the Digital Toys Inc. web application as user100@dt.com/user100. Navigate to Order History. You should now see the recently placed order in the Order History for this user.

Controller

Controllers determine the sequence in which the samplers are processed. These are analogous to the control flow constructs in common programming languages. The names of the controllers are indicative of their function.

Logical controllers can be categorized as follows:

1. Controllers for grouping
 - a. Simple Controller
 - b. Transaction Controller
2. Controllers for looping
 - a. Loop Controller
 - b. While Controller
 - c. ForEach Controller
3. Controllers for decision making
 - a. If Controller
 - b. Switch Controller
 - c. Once Only Controller
 - d. Interleave Controller
 - e. Random Controller
 - f. Random Order Controller

4. Controllers for modularity
 - a. Module Controller
 - b. Include Controller
5. Controllers for recording
 - a. Recording Controller
6. Other controllers
 - a. Critical Section Controller
 - b. Throughput Controller
 - c. Runtime Controller

JMeter can be extended by writing custom controllers.

Simple Controller

The *Simple Controller* provides no functionality beyond that of grouping, primarily to organize samplers and other logic controllers.

Note The Simple Controller is not often used. Instead, consider using the transaction controller, which is discussed later in the chapter.

Let's look at an example that illustrates the use of a Simple Controller.

Follow these steps or download [SimpleControllerTestPlan.jmx](#).¹²

1. Create a test plan and give it a meaningful name, such as **Simple Controller Test**.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Simple Controller.
5. Click on Simple Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
6. Click on Simple Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
7. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.

¹²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/SimpleControllerTestPlan.jmx

9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-33.

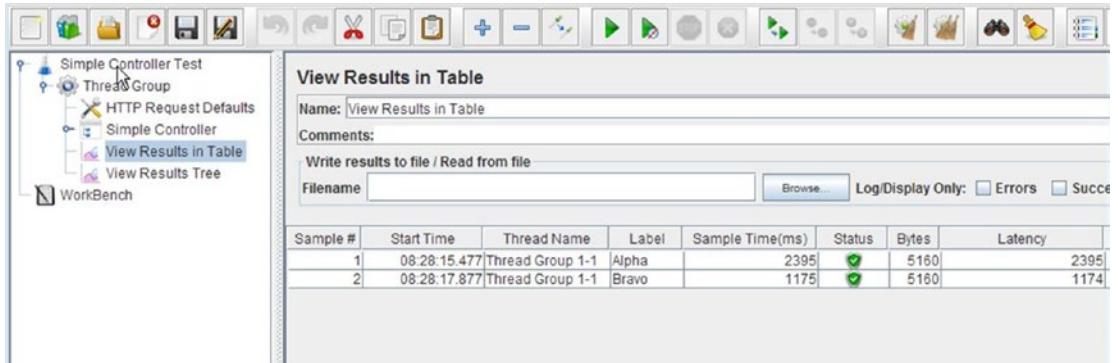


Figure 5-33. Simple Controller results

Transaction Controller

The *Transaction Controller* provides the functionality of grouping elements together, similar to Simple Controller. In addition to that, it generates an additional entry in the listener that measures the overall time taken to perform the nested test elements.

Let's look at an example that illustrates the use of a Transaction Controller.

Follow these steps or download [TransactionControllerTestPlan.jmx](#).¹³

1. Create a test plan and give it a meaningful name, such as **Transaction Controller Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to **Edit > Add > Logic Controller** and add Transaction Controller. Do not select any of the checkboxes.
5. Click on Transaction Controller and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
6. Click on Transaction Controller and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
7. Click on Thread Group and go to **Edit > Add > Listener**. Add View Results Tree.
8. Click on Thread Group and go to **Edit > Add > Listener**. Add View Results in Table.

¹³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/TransactionControllerTestPlan.jmx

9. Save the test plan.

10. Run the test.

The results will be similar to those shown in Figure 5-34.

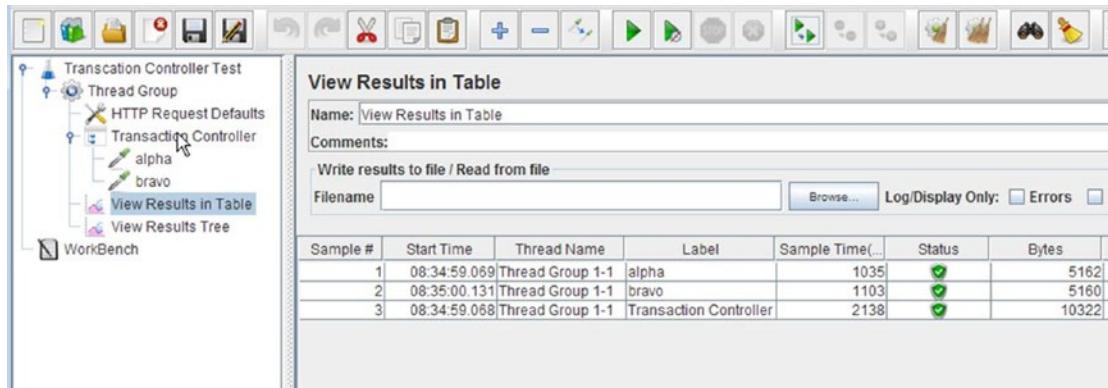


Figure 5-34. Transaction Controller

These results indicate that there is a new entry in the results table that has the combined aggregate values of the sample time (ms), latency, bytes, and connect time.

The Transaction Controller can function in *Parent Mode*. This can be enabled using a checkbox as shown in Figure 5-35 and then the additional sample is added as a parent of the nested samples.

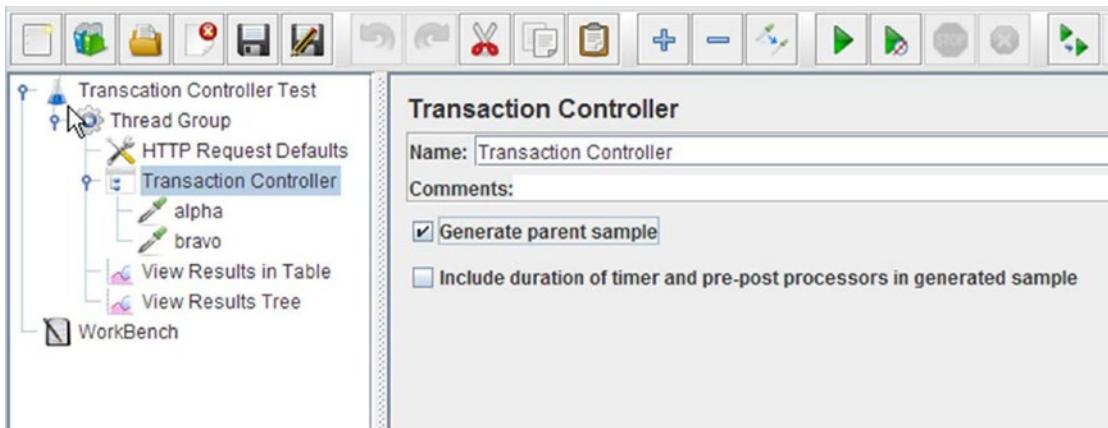


Figure 5-35. Transaction Controller parent mode

The Transaction Controller is considered successful only if all the samplers under it are successful.

In the Parent Mode, the individual samples can still appear in the View Results Tree, but no longer appear as separate entries in View Results in Table or in other listeners. Also, the sub-samples do not appear in CSV log files, but they can be saved to XML files (see Figure 5-36).

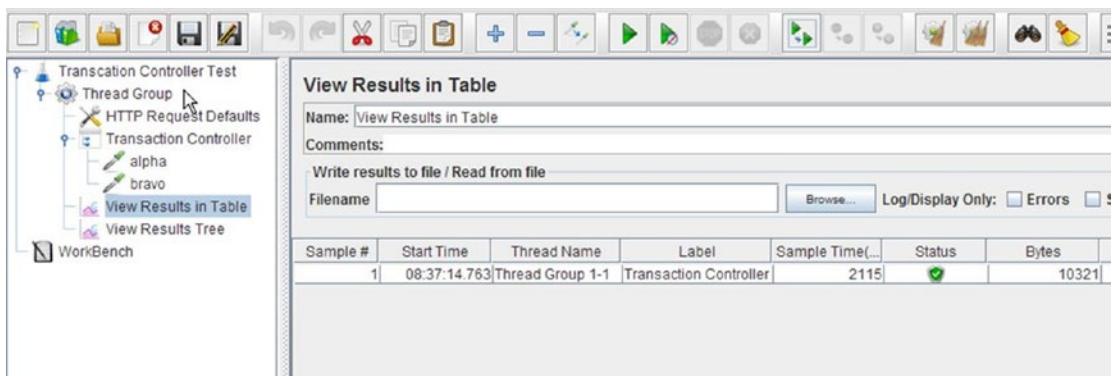


Figure 5-36. Transaction Controller parent mode results

By default, Transaction Controller does not include the time taken by timers and pre-processors. However, it can be configured to include the timers by enabling a checkbox, as shown.

Follow these steps or download `TransactionControllerConstantTimerTestPlan.jmx`.¹⁴

1. Open `TransactionControllerTestPlan.jmx`.
2. Go to File ▶ Save Test Plan as and type in a filename, such as `TransactionControllerConstantTimerTestPlan.jmx`.
3. Give the test plan a meaningful name, such as `Transaction Controller And Constant Timer Test`.
4. Click on Transaction Controller and go to Edit ▶ Add ▶ Timer. Add Constant Timer. Configure **Thread Delay (in Milliseconds)** to 1000 (see Figure 5-37).

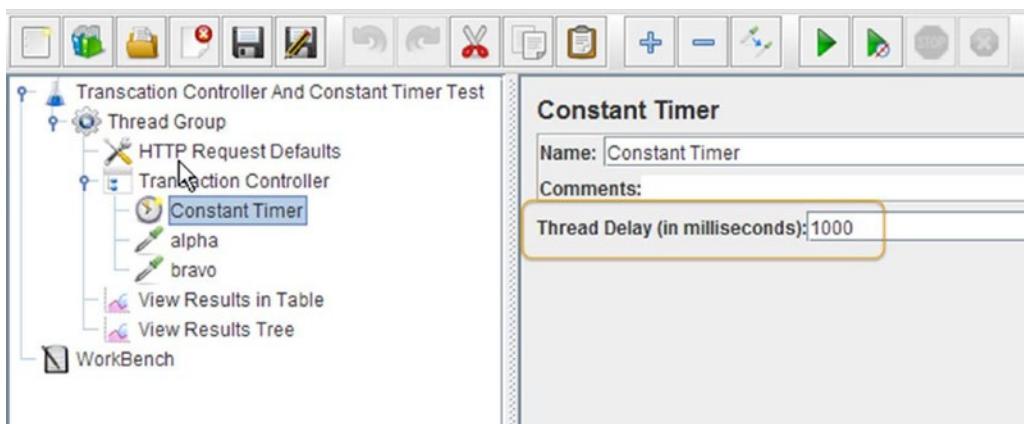


Figure 5-37. Constant timer

¹⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/TransactionControllerConstantTimerTestPlan.jmx

5. Save the test plan.
6. Run the test.

The results will be similar to those shown in Figure 5-38.

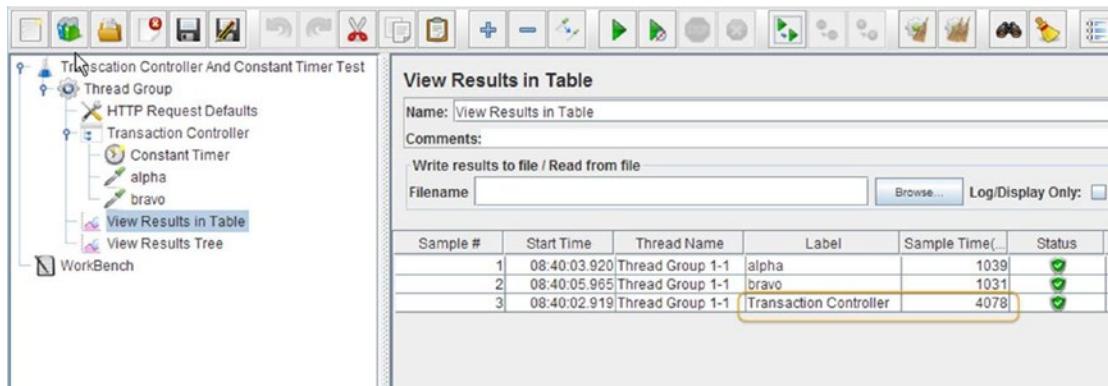


Figure 5-38. Consolidated entry of the Transaction Controller

The result indicates that the aggregate is one second more than the sum of the HTTP Request Sampler. This is because the time taken by the timer has been added to the aggregate.

Loop Controller

The *Loop Controller* provides a looping mechanism. It can repeat the execution of its nested elements a specified number of times. It also has a checkbox to configure it to loop forever.

Let's look at an example that illustrates the use of a Loop Controller.

Follow these steps or download `LoopControllerTestPlan.jmx`.¹⁵

1. Create a test plan and give it a meaningful name, such as **Loop Controller Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add HTTP Request Defaults. Configure **Server Name or IP** as `localhost` and **Port Number** as `8080`.
4. Click on Thread Group and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Name** as `Alpha`, **Path** as `/jmeter/alpha`, and **Method** as `GET`.
5. Click on Thread Group and go to **Edit > Add > Logic Controller**. Add Loop Controller. Configure the **Loop Count** as `4` (see Figure 5-39).

¹⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/LoopControllerTestPlan.jmx

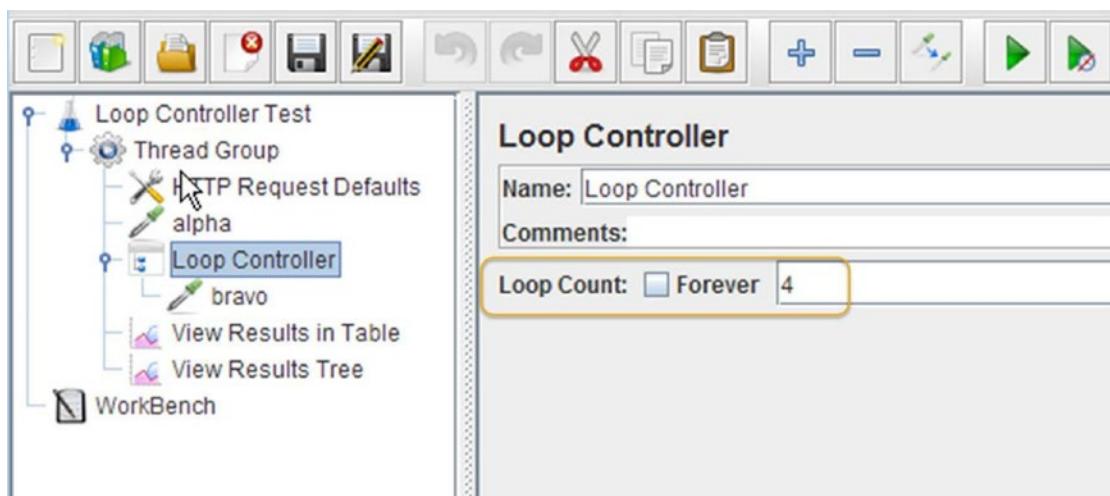


Figure 5-39. Loop Controller configuration

6. Click on Loop Controller and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as **Bravo**, **Path** as **/jmeter/bravo**, and **Method** as **GET**.
7. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
8. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-40.

Sample #	Start Time	Thread Name	Label	Sample Time(...)	Status	Bytes
1	08:42:05.482	Thread Group 1-1	alpha	1036	✓	5160
2	08:42:06.519	Thread Group 1-1	bravo	1055	✓	5161
3	08:42:07.576	Thread Group 1-1	bravo	1049	✓	5160
4	08:42:08.626	Thread Group 1-1	bravo	1070	✓	5161
5	08:42:09.697	Thread Group 1-1	bravo	1029	✓	5161

Figure 5-40. Loop Controller results

As you can see in the results, alpha was called once followed by four requests to bravo.

Note This controller is used to simulate high traffic to certain pages.

The Loop Controller can be configured to loop indefinitely by enabling the Forever checkbox. This gets enabled even if you enter any negative number for Loop Count.

If you run the test with the Forever checkbox enabled, you need to click on the Shutdown button and terminate the test gracefully. Clicking on the Stop button terminates the threads abruptly, and you may see some errors in the requests.

Runtime Controller

The *Runtime Controller* controls the duration for which its child elements are run.

A Runtime Controller executes its child elements in its hierarchy for the specified duration. At the end of the nested elements, it loops through again. Using the same logic, if the specified time runs out, the Runtime Controller stops execution even if it has executed only a part of the nested elements. The currently executing element is allowed to complete, but the newer elements are not executed once the specified time is over.

The configuration is very simple. You just have to enter the number of milliseconds for which the child elements should be run. If the duration field is 0 or a negative number, all the child elements are completely skipped.

This also works as a looping mechanism; the only difference being that instead of the specified number of iterations, the child elements are looped for a specified elapsed time.

Let's look at an example that illustrates the use of a Runtime Controller.

Follow these steps or download `RuntimeControllerTestPlan.jmx`.¹⁶

1. Create a test plan and give it a meaningful name, such as Runtime Controller Test
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** to localhost and **Port Number** to 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Runtime Controller. Configure **Runtime (seconds)** to 10 (see Figure 5-41).

¹⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/RuntimeControllerTestPlan.jmx

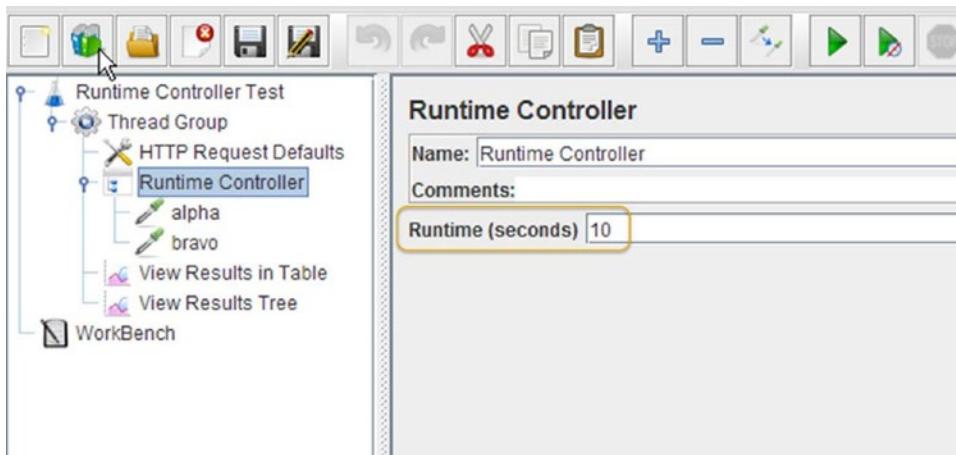


Figure 5-41. Runtime Controller configuration

5. Click on Runtime Controller and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
6. Click on Runtime Controller and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
7. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
8. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-42.

Sample #	Start Time	Thread Name	Label	Sample Time(...)	Status	Bytes
1	08:44:03.590	Thread Group 1-1	alpha	1099	✓	5161
2	08:44:04.695	Thread Group 1-1	bravo	1033	✓	5162
3	08:44:05.729	Thread Group 1-1	alpha	1072	✓	5160
4	08:44:06.802	Thread Group 1-1	bravo	1031	✓	5161
5	08:44:07.837	Thread Group 1-1	alpha	1092	✓	5161
6	08:44:08.932	Thread Group 1-1	bravo	1107	✓	5162
7	08:44:10.041	Thread Group 1-1	alpha	1043	✓	5162
8	08:44:11.085	Thread Group 1-1	bravo	1032	✓	5161
9	08:44:12.129	Thread Group 1-1	alpha	1031	✓	5161
10	08:44:13.161	Thread Group 1-1	bravo	1050	✓	5161

Figure 5-42. Runtime Controller execution duration

Observe that the time difference between the last and the first sample is 9.182 seconds, which is approximately 10 seconds. This is working as configured. See Table 5-1 for more detail.

Table 5-1. Difference Between Samples

Time of Last Sample	Time of First Sample	Difference
08:44:13:161	08:44:03:590	00:00:09:57

Throughput Controller

The Throughput Controller controls the number of executions of its child elements. This is a misnomer, as it does not control the throughput; use the Constant Throughput Timer for that.

You can configure this in two modes:

- **Number of Executions:** The child elements are executed until the specified count is reached and the subsequent executions are skipped. To configure this, choose Total Executions and specify a value for Throughput.
- **Percentage of Executions:** The concept is the same except that the number of executions of the child elements is restricted by the percentage configured. To configure this, choose Percent Executions and specify a value for Throughput (omit the % sign).

Both of these modes limit the number of executions of the controller. These limits apply collectively to all the threads in the thread group. These limits apply on a per thread basis when the Per User checkbox is enabled.

The following example demonstrates this concept. We will configure the Throughput Controller in Total Executions mode to execute based on a count specified.

Follow these steps or download [ThroughputControllerTestPlan.jmx](#).¹⁷

1. Create a test plan and give it a meaningful name, such as Throughput Controller Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Loop Count** as 10.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
5. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Throughput Controller. Configure **Throughput** as 3, and then select **Total Executions** from the drop-down (see Figure 5-43).

¹⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/ThroughputControllerTestPlan.jmx

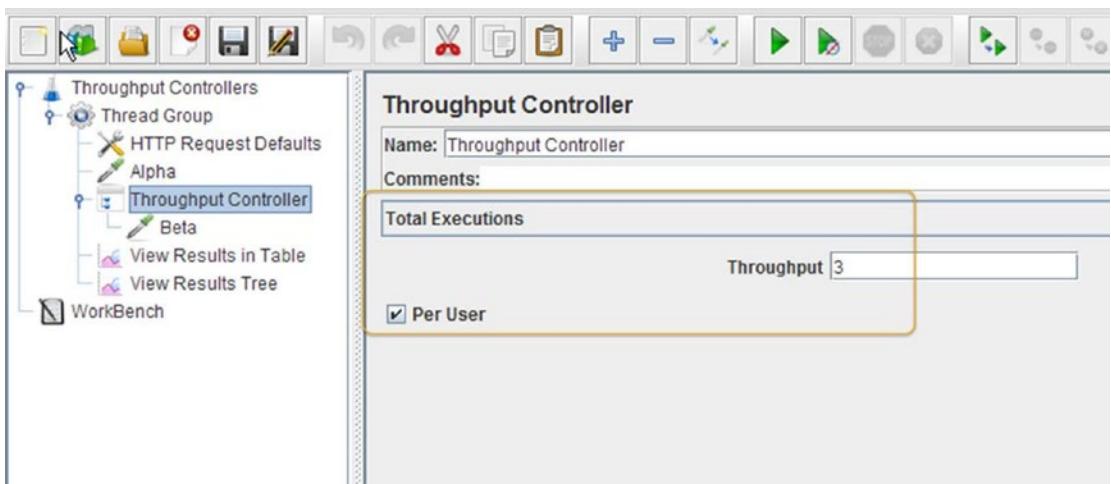


Figure 5-43. Throughput Controller configuration

6. Click on Throughput Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as beta, **Path** as /jmeter/beta, and **Method** as GET.
7. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-44.

The screenshot shows the JMeter interface with the 'View Results in Table' tree node selected. A 'View Results in Table' dialog is open on the right, titled 'View Results in Table'. It contains fields for 'Name' (set to 'View Results in Table'), 'Comments', and options to 'Write results to file / Read from file'. A 'Filename' input field and 'Browse...' button are present, along with checkboxes for 'Log/Display Only', 'Errors', and 'Summary'. Below the dialog is a table showing test results. The table has columns for Sample #, Start Time, Thread Name, Label, Sample Time(ms), Status, and Bytes. The data is as follows:

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	08:46:09.176	Thread Group 1-1	Alpha	1308	✓	5160
2	08:46:10.488	Thread Group 1-1	Beta	1029	✓	5160
3	08:46:11.518	Thread Group 1-1	Alpha	1040	✓	5160
4	08:46:12.559	Thread Group 1-1	Beta	1033	✓	5158
5	08:46:13.594	Thread Group 1-1	Alpha	1025	✓	5161
6	08:46:14.621	Thread Group 1-1	Beta	1036	✓	5159
7	08:46:15.658	Thread Group 1-1	Alpha	1043	✓	5161
8	08:46:16.703	Thread Group 1-1	Alpha	1025	✓	5161
9	08:46:17.729	Thread Group 1-1	Alpha	1021	✓	5162
10	08:46:18.751	Thread Group 1-1	Alpha	1027	✓	5161
11	08:46:19.780	Thread Group 1-1	Alpha	1023	✓	5162
12	08:46:20.805	Thread Group 1-1	Alpha	1050	✓	5160
13	08:46:21.856	Thread Group 1-1	Alpha	1029	✓	5160

Figure 5-44. Throughput Controller results

The `/jmeter/beta` HTTP request, being the child element of the Throughput Controller, was executed only three times, as configured in the Throughput count. The `/jmeter/alpha` HTTP request, being outside of the Throughput Controller, was executed for a full 10 times, as configured in the thread group's Loop Count.

The various combinations of configurations are summarized in Table 5-2.

Table 5-2. Configuration Summary

Threads (Users)	Loop Count	Per User	Mode	Throughput	Req Count (<code>/jmeter/alpha</code>)	Req. Count (<code>/jmeter/beta</code>)
1	10	No	Count	3	10	3
1	10	No	Percent	20 %	10	2
2	10	No	Count	3	20	3
2	10	No	Percent	20 %	20	4
2	10	Yes	Count	3	20	6
2	10	Yes	Percent	20 %	20	4

`/jmeter/alpha` is a direct child element of the thread group and is outside the scope of the Throughput Controller. The number of requests to the `/jmeter/alpha` is always determined by the Number of Threads (users) times the Loop Count configured in the thread group.

`/jmeter/beta` is a child element under Throughput Controller. The number of requests to the `/jmeter/beta` is restricted by the Throughput Controller configuration.

When the Per User flag is enabled and the Throughput Controller is in Total Execution mode, the number of times the child elements are executed is the Number of Threads (Users) times the Throughput.

When the Throughput Controller is in Percent mode, the result happens to be the same whether or not the Per User checkbox is enabled. This does not change the execution count when Throughput Controller is in Percent mode.

Once Only Controller

Once Only Controller executes its child elements only once per thread. This is typically used to perform logins or another use-case that's needed only once for a user session.

There is no configuration for this Once Only Controller. The Once Only Controller should be a child element of the thread group or Loop Controller. Otherwise, the behavior is not defined.

The following is an example of using Once Only Controller as a child of thread group, where thread group has been configured to do the looping.

Follow these steps or download `OnceOnlyControllerTestPlan.jmx`.¹⁸

1. Create a test plan and give it a meaningful name, such as Once Only Controller Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (Users)** as 2 and **Loop Count** as 4.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.

¹⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/OnceOnlyControllerTestPlan.jmx

4. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Once Only Controller. It has no configuration.
5. Click on Once Only Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Alpha, Path as /jmeter/alpha, and **Method** as GET.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
7. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Charlie, Path as /jmeter/charlie, and **Method** as GET.
8. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Delta, Path as /jmeter/delta, and **Method** as GET.
9. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
10. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
11. Save the test plan.
12. Run the test.

The results will be similar to those shown in Figure 5-45.

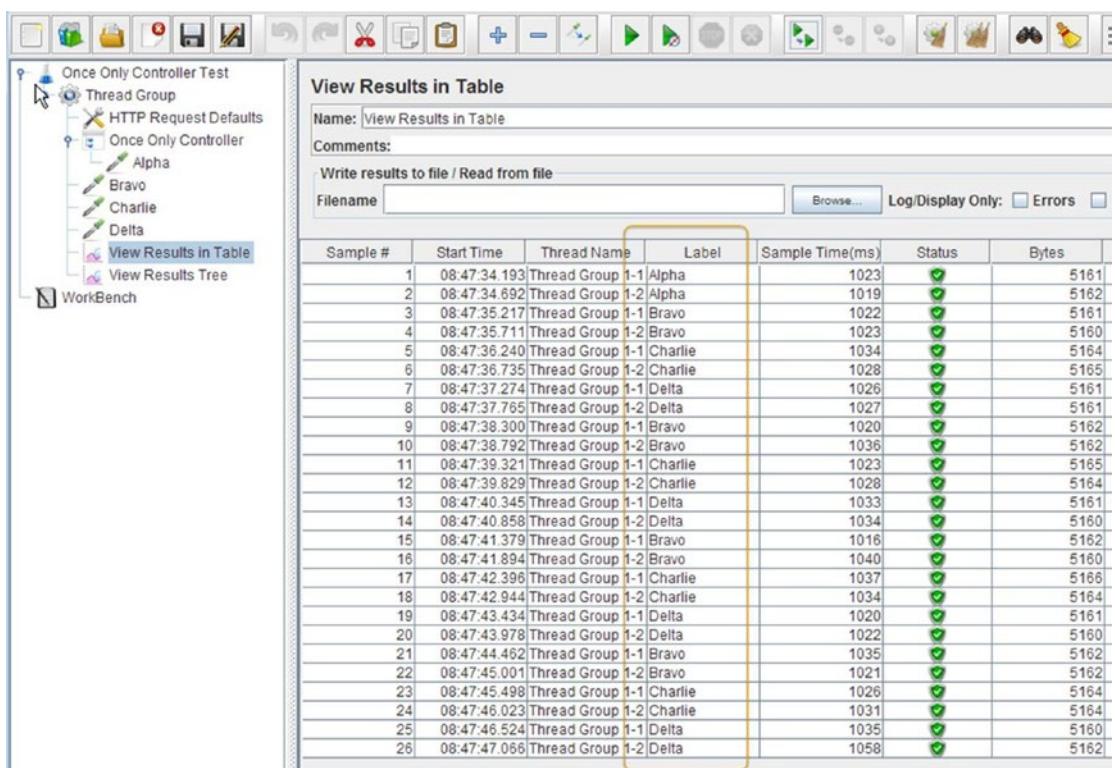


Figure 5-45. Once Only Controller with only one request to alpha

These results indicate that the HTTP request for /jmeter/alpha, the child element of the Once Only Controller, was executed only two times, which is once per thread, as configured. The HTTP requests for /jmeter;bravo, /jmeter;charlie, and /jmeter;delta were executed eight times as the Number of Threads (Users) was 2 and the Loop Count was 4.

Interleave Controller

The Interleave Controller executes only one of its child elements per loop iteration. Each time it iterates, it picks the next child element in sequence.

A child controller is considered a sub-controller. By default, a sub-controller, including all its children, is treated as a single unit. If the Ignore Sub-Controller Blocks checkbox is enabled, the grouping implied by the sub-controller is ignored and the child elements of the sub-controller are treated as the direct child elements of the Interleave Controller.

The Interleave Controller is used to distribute the requests among a set of URLs. Let's look at an example that illustrates the use of the Interleave Controller.

Follow these steps or download `InterleaveAndLoopControllerTestPlan.jmx`.¹⁹

1. Create a test plan and give it a meaningful name, such as `Interleave And Loop Controller Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Config Element`. Add HTTP Request Defaults. Configure **Server Name or IP** as `localhost` and **Port Number** as `8080`.
4. Click on Thread Group and go to `Edit > Add > Logic Controller`. Add Loop Controller. Configure **Loop Count** as `4`.
5. Click on Loop Controller and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Name** as `Zulu`, **Path** as `/jmeter/zulu`, and **Method** as `GET`.
6. Click on HTTP Request Zulu and go to `Edit > Add > Logic Controller`. Add Interleave Controller. Leave the checkbox titled **Ignore Sub-Controller Blocks** unchecked (see Figure 5-46).

¹⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/InterleaveAndLoopControllerTestPlan.jmx

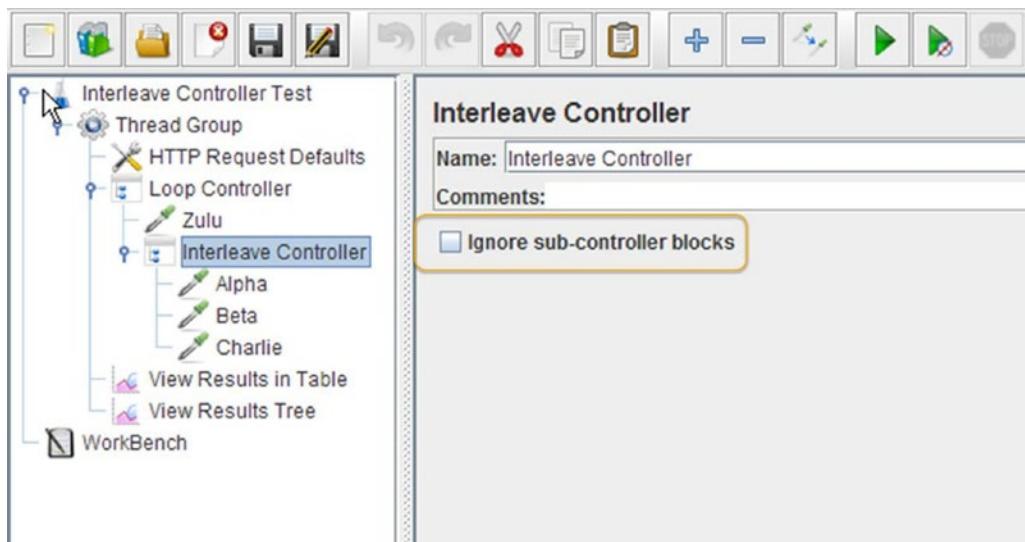


Figure 5-46. Interleave Controller configuration

7. Click on Interleave Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
8. Click on Interleave Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Beta, **Path** as /jmeter/beta, and **Method** as GET.
9. Click on Interleave Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Charlie, **Path** as /jmeter/charlie, and **Method** as GET.
10. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
11. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
12. Save the test plan.
13. Run the test.

The results will be similar to those shown in Figure 5-47.

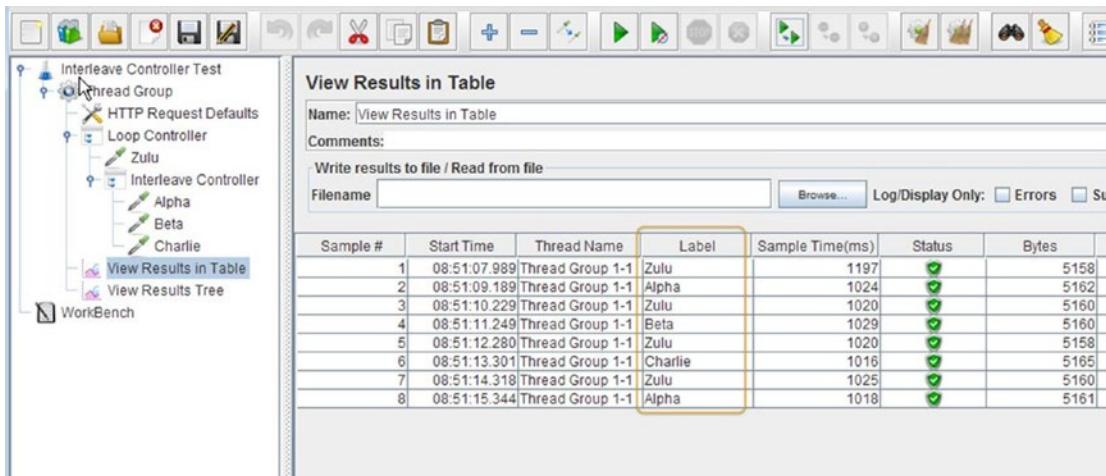


Figure 5-47. Interleave Controller alternative requests

These results indicate that the Interleave Controller cycles the requests between Alpha, Beta, and Charlie for each loop iteration.

The Interleave Controller alternates the requests separately for each thread. You can verify this by modifying this example in the following steps.

1. Modify the thread group configuration and update **Number of Threads (Users)** field to 2.
2. Run the test.

The results will be similar to those shown in Figure 5-48.

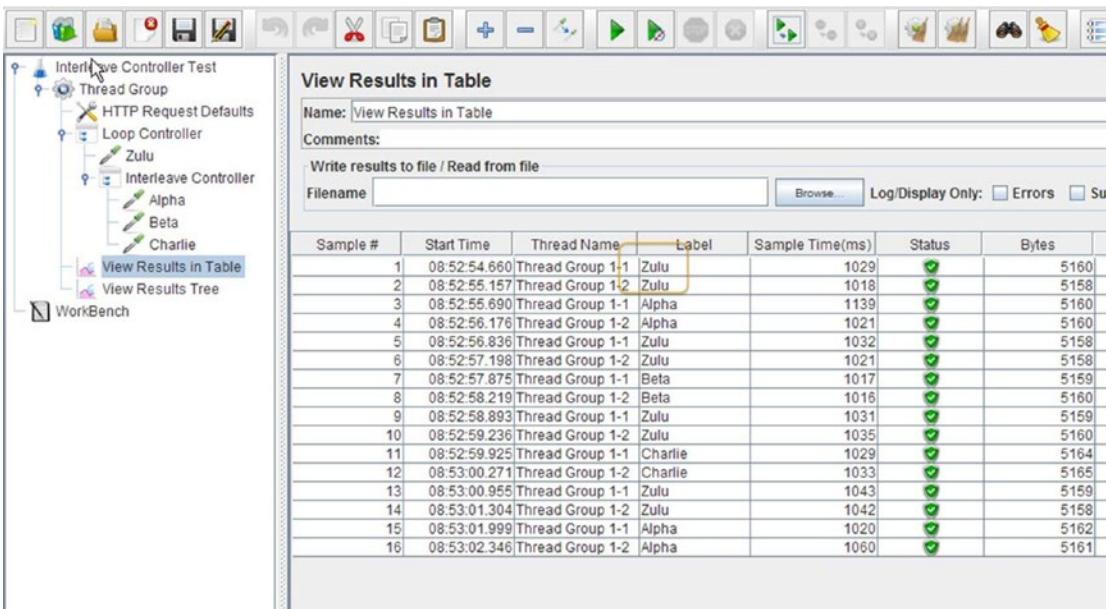


Figure 5-48. Interleave Controller updated results

In these results, the Interleave Controller alternates the requests for each thread. That is why we see two requests for Zulu, Alpha, Zulu, Beta, and so on.

Let's look at the effect of having controllers as child elements of an Interleave Controller.

Follow these steps or download [InterleaveAndSimpleControllerTestPlan.jmx](#).²⁰

1. Open [InterleaveAndLoopControllerTestPlan.jmx](#).
2. Go to File ► Save Test Plan As and give it a meaningful name, such as **Interleave And Simple Controller Test**.
3. Click on Interleave Controller and go to Edit ► Add ► Logic Controller. Add Simple Controller.
4. Select Beta and Charlie HTTP requests and drag these requests under the Simple Controller as child elements.
5. Save the test plan.
6. Run the test.

The results will be similar to those shown in Figure 5-49.

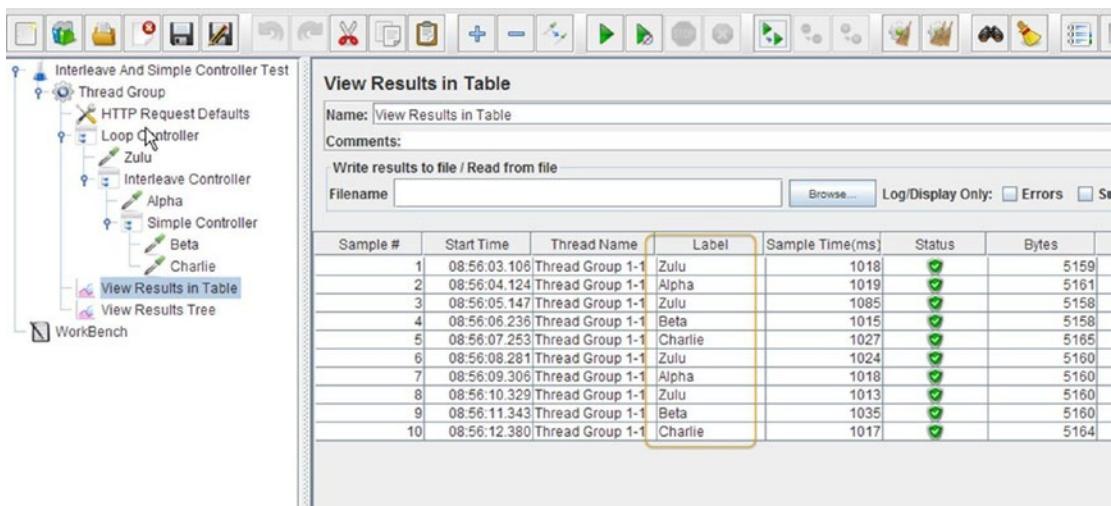


Figure 5-49. Interleave Controller updated results

These results indicate that the Interleave Controller alternates requests between its two child elements—Alpha and the Simple Controller. All the elements inside the Simple Controller are treated as a unit. So the request sequence is Zulu, Alpha, Zulu, Beta, Charlie.

Let's enable the checkbox and see how the request is distributed.

Follow these steps or download [InterleaveWithIgnoreSubControllersTestPlan.jmx](#).²¹

²⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/InterleaveAndSimpleControllerTestPlan.jmx

²¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/InterleaveWithIgnoreSubControllersTestPlan.jmx

1. Open InterleaveAndSimpleControllerTestPlan.jmx.
2. Go to File ► Save Test Plan as and type in a filename, such as InterleaveWithIgnoreSubControllersTestPlan.jmx.
3. Give the test plan a meaningful name, such as Interleave Controller With Ignore Sub Controllers.
4. Click on Interleave Controller and enable the checkbox **Ignore Sub-Controller Blocks** (see Figure 5-50).

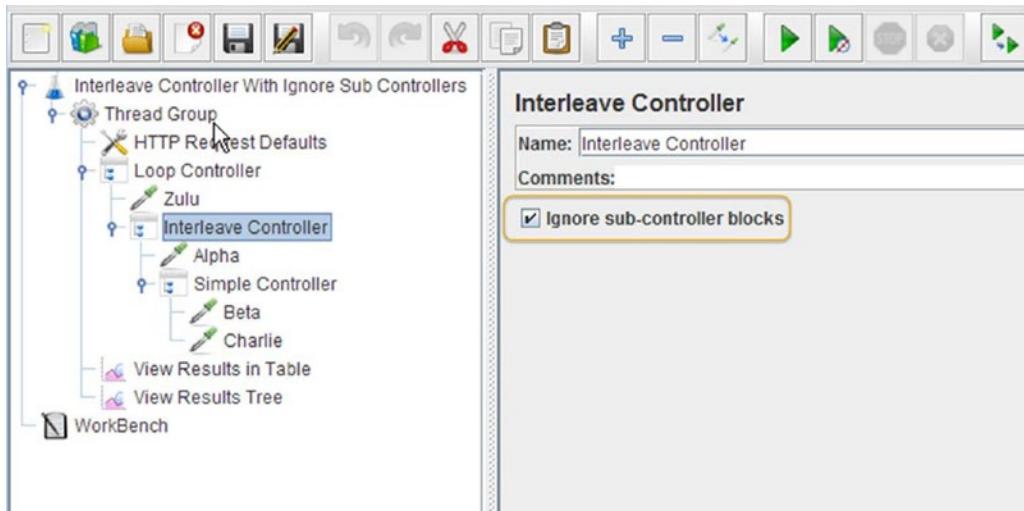


Figure 5-50. Interleave Controller ignore sub-controller block configuration

5. Save the test plan.
6. Run the test.

The results will be similar to those shown in Figure 5-51.

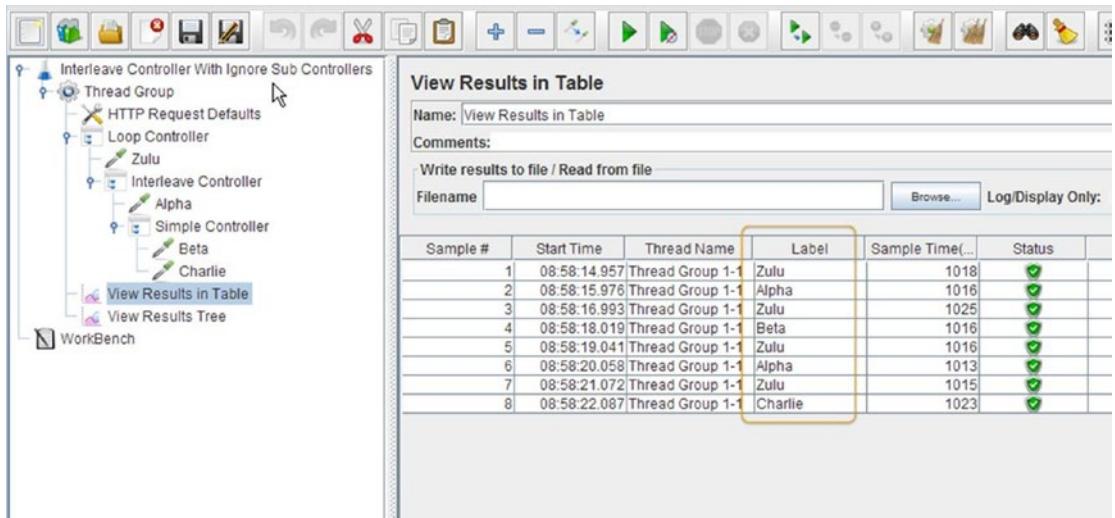


Figure 5-51. Interleave Controller alternate requests

These results indicate that the elements under the Simple Controller are also getting interleaved. It has changed the request sequence to Zulu, Alpha, Zulu, Beta, Zulu, Alpha, Zulu, Charlie.

Random Controller

The Random Controller is similar to the Interleave Controller except that the order of interleaving is random instead of sequential. The configuration is just like the Interleave Controller.

In the previous tests, replace Interleave Controller with Random Controller, run the tests, and then verify the results.

Random Order Controller

The Random Order Controller executes all its child elements but in random order. There is no other configuration for this controller.

Follow these steps or download `RandomOrderControllerTestPlan.jmx`.²²

1. Create a test plan and give it a meaningful name, such as Random Order Controller Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Random Order Controller.

²²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/RandomOrderControllerTestPlan.jmx

5. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
 6. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
 7. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Charlie, **Path** as /jmeter/charlie, and **Method** as GET.
 8. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Name** as Zulu, **Path** as /jmeter/zulu, and **Method** as GET.
 9. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
 10. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
 11. Save the test plan.
 12. Run the test.

The results will be similar to those shown in Figure 5-52.

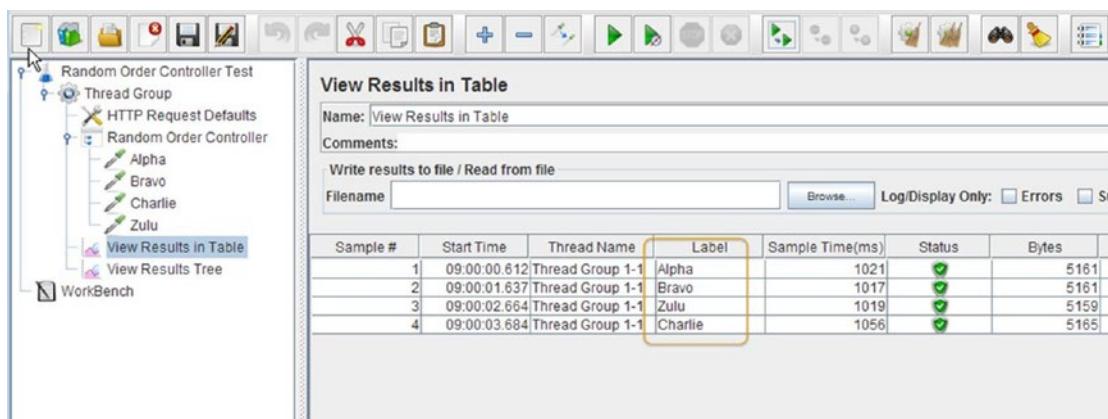


Figure 5-52. Random Order Controller results

These results indicate that all the child samplers were executed in random order.

Switch Controller

The Switch Controller is analogous to the switch/case programming construct. The Switch Controller executes only one of its child elements after matching the element's name with the configured Switch value. If the Switch value is an integer, it executes the child element based on the sequence number.

Note The sequence number starts at 0.

Let's look at an example.

Follow these steps or download `SwitchControllerTestPlan.jmx`.²³

1. Create a test plan and give it a meaningful name, such as `Switch Controller Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Config Element`. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to `Edit > Add > Logic Controller`. Add Switch Controller. Configure **Switch Value** as Bravo (see Figure 5-53).

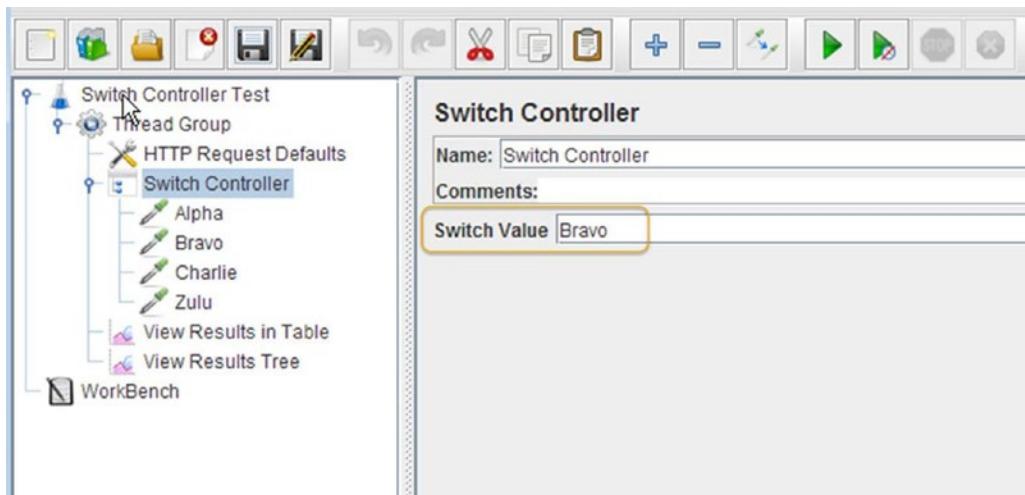


Figure 5-53. Switch Controller configuration

5. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
6. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
7. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Name** as Charlie, **Path** as /jmeter/charlie, and **Method** as GET.
8. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Name** as Zulu, **Path** as /jmeter/zulu, and **Method** as GET.
9. Click on Thread Group and go to `Edit > Add > Listener`. Add View Results Tree.
10. Click on Thread Group and go to `Edit > Add > Listener`. Add View Results in Table.

²³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/SwitchControllerTestPlan.jmx

11. Save the test plan.
12. Run the test.

The results will be similar to those shown in Figure 5-54.

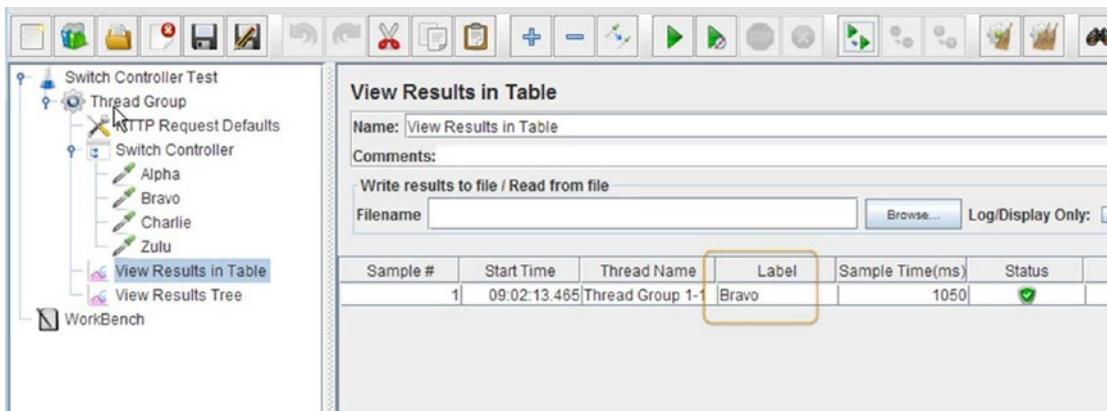


Figure 5-54. Switch Controller results

In these results, we can see that the Switch Controller has matched the Switch value Bravo with its child of the same name. So the request goes to the HTTP request Bravo.

Let's see how the Switch Controller behaves when configured with a number.

1. In the previous test plan, click on **Switch Controller** and modify the **Switch value** to 3.
2. Run the test. The results will be similar to those shown in Figure 5-55.

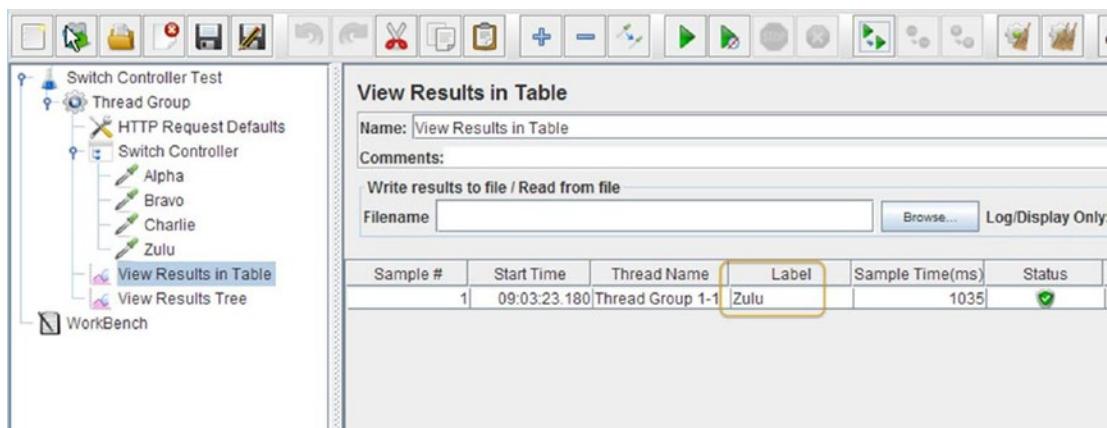


Figure 5-55. Switch Controller results based on number

The results indicate that the Switch Controller picked the fourth child element as the Switch value was 3. (Remember that the numbering starts with 0.)

ForEach Controller

The ForEach Controller has one or more child elements over which it iterates. It can be configured with the following parameters. For each iteration, the ForEach Controller performs the following:

- Forms a sequence number by incrementing the Start Index for loop (Exclusive) option.
- Forms the name of a user defined variable by concatenating the Input Variable Prefix, “_” and the sequence number.
- Sets the Output Variable Name to the value obtained by looking up the user defined variable. This Output Variable Name is then available to the child elements.

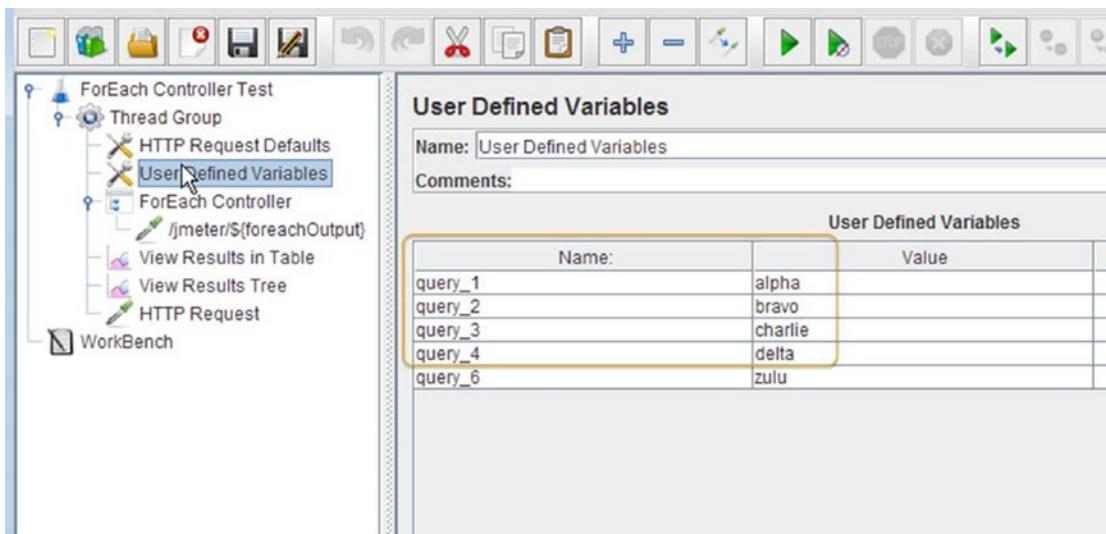
The number of iterations is equal to the difference between the Start index and End index. If the Start index and the End index have not been specified, JMeter can figure those out by looking at the user defined variables, starting with the Input Variable Prefix string. Such variables need to be in numerical sequence. Any break in the sequence will cause ForEach Controller to finish.

If the Add “_” Before Number? checkbox is not checked, then “_” is not used in forming the name of the user defined variable. Let's look at an example.

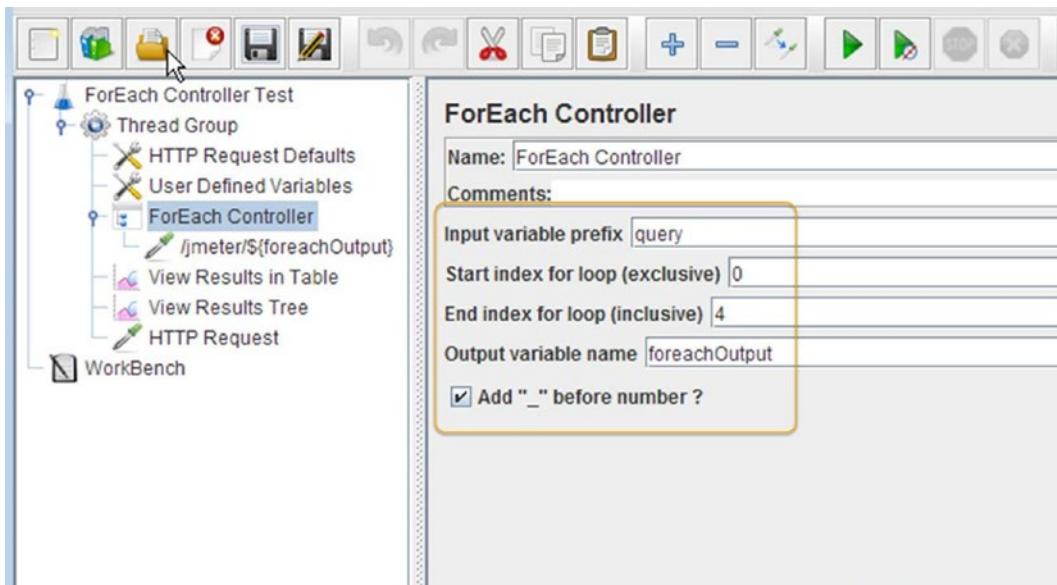
Follow these steps or download `ForEachControllerTestPlan.jmx`.²⁴

1. Create a test plan and give it a meaningful name, such as **ForEach Controller Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to **Edit > Add > Config Element**. Add User Defined Variables. Configure **User Defined Variables** add **Name/Value** pairs as query_1/"alpha", query_2/"bravo", query_3/"charlie", and query_4/"delta" (see Figure 5-56).

²⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/ForEachControllerTestPlan.jmx

**Figure 5-56.** User defined variables

5. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add ForEach Controller. Configure **Input Variable Prefix** as query, **Start Index for Loop (Exclusive)** as 0, **End Index for Loop (Inclusive)** as 4, and **Output Variable Name** as foreachOutput. Enable the **Add "_" before number?** checkbox (see Figure 5-57).

**Figure 5-57.** ForEach Controller configuration

6. Click on ForEach Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as /jmeter/\${foreachOutput}, **Path** as /jmeter/\${foreachOutput}, and **Method** as GET. Note that we are referencing the output variable defined in the ForEach Controller.
7. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-58.

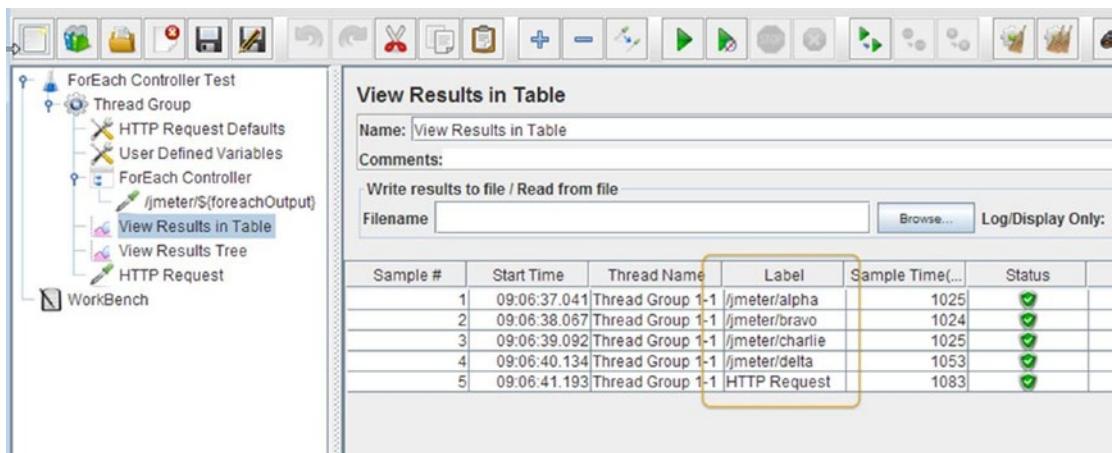


Figure 5-58. ForEach Controller subsequent request

These results indicate that the ForEach Controller has set `foreachOutput` successively to `alpha`, `bravo`, `charlie`, and `delta`. This has resulted in the `HTTP Request Sampler`, configured with the URL of `/jmeter/${foreachOutput}`, being invoked with `/jmeter/alpha`, `/jmeter/bravo`, `/jmeter/charlie`, and `/jmeter/delta`, respectively.

Caution Input Variable Prefix and Output Variable Name are required parameters. If you omit them, there is no error checking, not even a log message. JMeter will simply stop execution without warning.

If Controller

The If Controller is useful for decision/branching logic. The configuration is simple, with only two checkboxes.

The Interpret Condition as Variable Expression? checkbox indicates whether the expression is evaluated as a JavaScript expression (the default) or as a variable expression (compared with the string "true").

The Evaluate for All Children? checkbox indicates whether the condition should be evaluated before processing each of the child elements. If it's not checked, the condition is evaluated only when it is encountered for the first time.

Let's look at an example by using user defined variables.

Follow these steps or download IfControllerUserDefinedVariableTestPlan.jmx.²⁵

1. Create a test plan and give it a meaningful name, such as If Controller User Defined Variable Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add User Defined Variables. Configure **User Defined Variables** add **Name/Value** pair as myUserVar/alpha.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Alpha, **Path** as /jmeter/alpha, and **Method** as GET.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Select **JMeter Properties** as False, **JMeter Variables** as True, and **System Properties** as False.
7. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add If Controller. Configure **Condition (Default JavaScript)** as "\${myUserVar}" == "alpha". Leave the other checkboxes unchecked (see Figure 5-59).

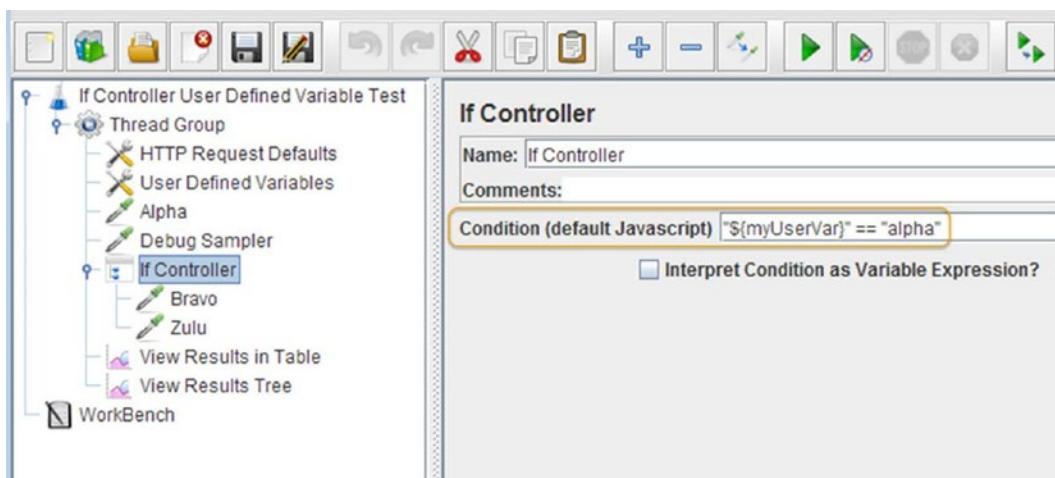


Figure 5-59. If Controller configuration

²⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/IfControllerUserDefinedVariableTestPlan.jmx

8. Click on If Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Bravo, **Path** as /jmeter;bravo, and **Method** as GET.
9. Click on If Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Name** as Zulu, **Path** as /jmeter/zulu, and **Method** as GET.
10. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
11. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
12. Save the test plan.
13. Run the test.

The results will be similar to those shown in Figure 5-60.

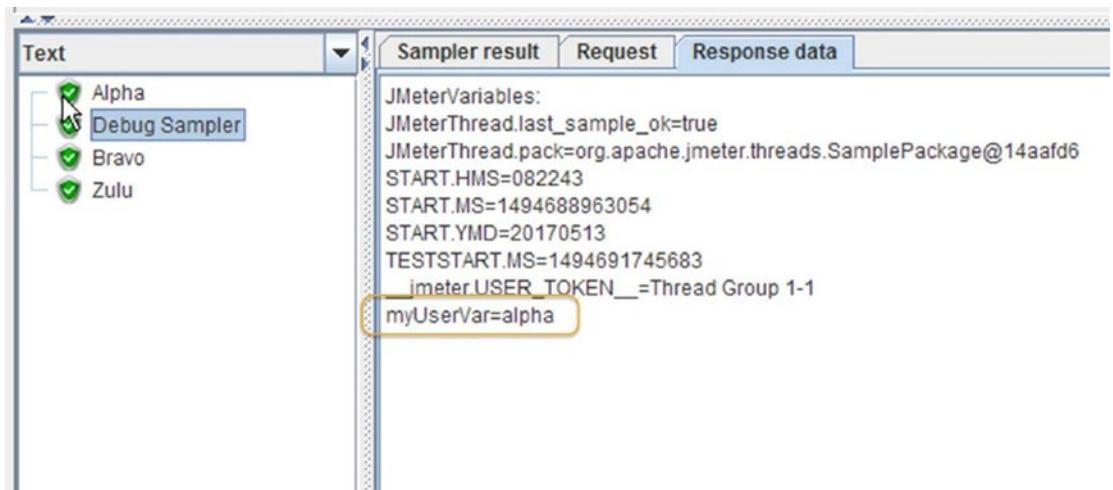


Figure 5-60. If Controller results

Click on View Results Tree and then Debug Sampler and then choose the Response Data tab. You will see that the user defined variable `myUserVar` is defined as `alpha`.

These results indicate that the If Controller has evaluated the expression "`{myUserVar}`" == "alpha" as a JavaScript expression and found it to be true, and this resulted in the execution of the child elements bravo and zulu (see Figure 5-61).

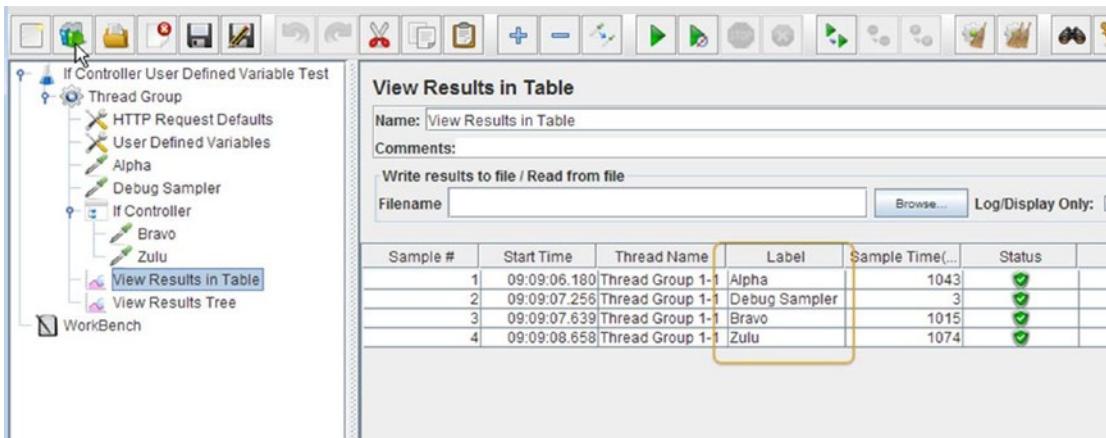


Figure 5-61. If Controller condition as true

Caution If the condition expression has a syntax error or if the variable is not found, JMeter will simply stop execution without any error popping up. If you have not selected the Interpret Condition as Variable Expression flag, a few DEBUG logs are generated.

The condition is interpreted as a JavaScript expression (the default) that evaluates to true or false. If the Interpret Condition as Variable Expression checkbox is enabled, then you can use other expressions.

If you enable the Interpret Condition as Variable Expression checkbox and replace the JavaScript condition "`{myUserVar}`" == "alpha" with \${__jexl("{myUserVar}" == "alpha")}, it will produce the same behavior.

In the previous examples, you used user defined variables to define the conditions in test scripts that were statically defined. In the following example, let's see how you can use variables extracted from the responses.

Follow these steps or download `IfControllerDynamicVariableTestPlan.jmx`.²⁶

1. Open `IfControllerUserDefinedVariableTestPlan.jmx`.
2. Click on User Defined Variables and go to Edit ▶ Remove.
3. Click on Thread Group and go to Edit ▶ Add ▶ Post Processor. Add CSS/JQuery Extractor. Configure **CSS/JQuery Expression** as title (a representation of the JQuery selector of `$(".title")`) and **Reference Name** as `myDynVar`. This is the name of the variable that will be set and available for the If Controller.
4. Add a Debug Sampler after each of the HTTP requests. So you should have three Debug Samplers, one after each of the HTTP requests, alpha, bravo, and zulu. This will allow you to examine the variables after the execution of every sampler.

²⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/IfControllerDynamicVariableTestPlan.jmx

5. Configure the If Controller with the condition "\${myDynVar}" == "alpha". Do not enable any other checkboxes.
6. Save the test plan.
7. Run the test.

The results will be similar to those shown in Figure 5-62.

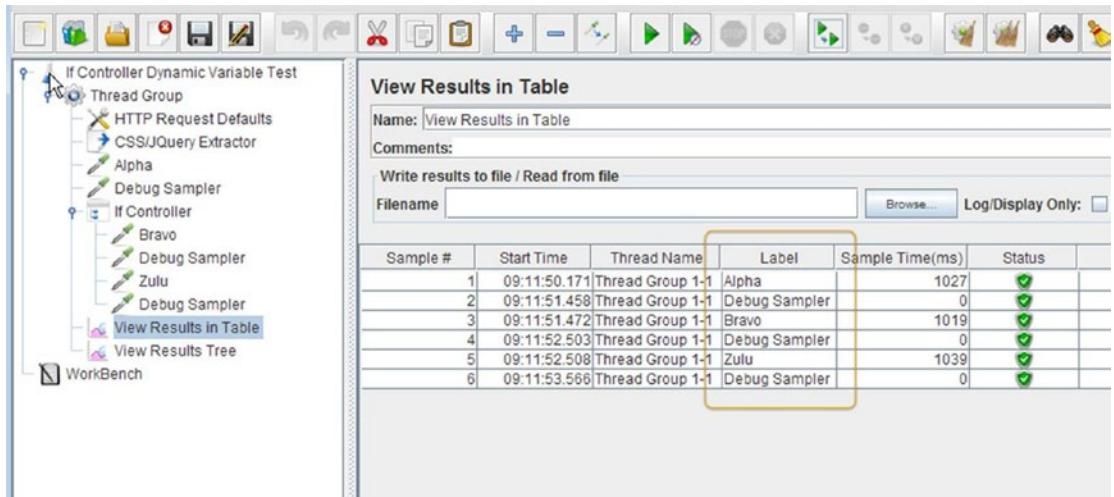


Figure 5-62. If Controller dynamic variable results

8. Click on View Results Tree and then on Debug Sampler. Click on the Response Data tab to check the variable myDynVar; you will find that it has been assigned to the title of the page. The values were changed to bravo and zulu after further requests were executed.

The If Controller was placed right after the HTTP sampler alpha. So the variable was set to alpha. The If Controller evaluated the condition and compared this variable to the value of alpha and found it to be true. So the child elements bravo and zulu were executed.

Note Do not enclose the JQuery selector using \$().

Now let's look at the functionality of the Evaluate for All Children checkbox. As the title indicates, selecting this checkbox causes the If Controller to evaluate the condition before every child sampler is run.

Let's illustrate this using the following example.

Follow these steps or download [IfControllerDynamicVariableEvaluateChildTestPlan.jmx](#).²⁷

1. Open IfControllerUserDefinedVariableTestPlan.jmx.
2. Click on the If Controller, modify the configuration, and enable the **Evaluate for All Children?** checkbox.

²⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/controllers/IfControllerDynamicVariableEvaluateChildTestPlan.jmx

3. Run the test.

The results will be similar to those shown in Figure 5-63.

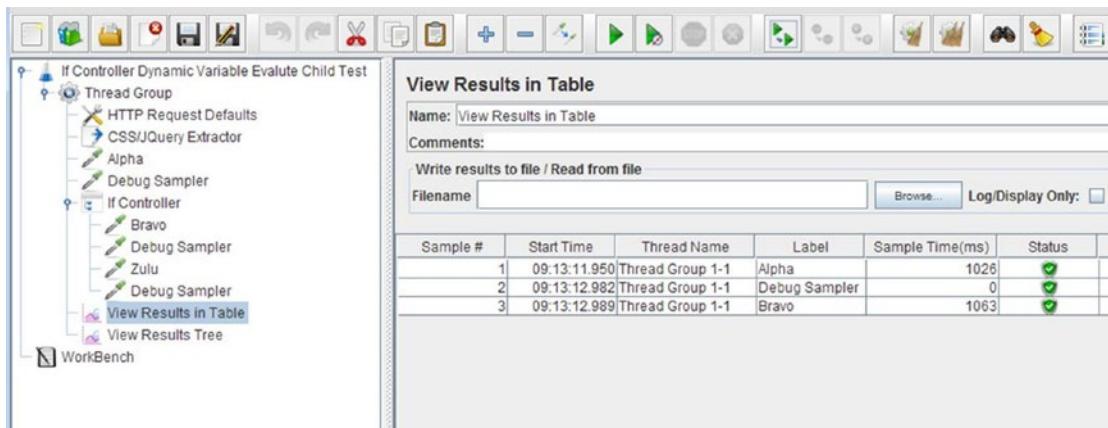


Figure 5-63. If Controller results evaluate child

- Click on View Results Tree and then on Debug Sampler. Click on the Response Data tab and check the value of the variable myDynVar; you will find that it has been assigned to the title of the page.

The If Controller was placed right after the HTTP request alpha. So the variable has been set to alpha. The If Controller evaluated the condition and compared this variable to the value of alpha and found it to be true. So the child element bravo was executed.

Since the Evaluate for All Children? checkbox was enabled, the condition is evaluated again before execution of second child element zulu. But the myDynVar is set to bravo, which is the title of the page that bravo returned. So the condition check fails and the second child element zulu is not executed.

Timers

When users view a web page, they do not click on every link on that page without reason. The user first has to read the web page, understand it, and then decide on the next course of action. This process varies from user to user and from web page to web page. Some web pages require that the user enter input or select files to upload, which introduces delays. You can simulate these delays by using *timers*.

The duration of the delay varies highly depending on the context and the user. To obtain a feel for the kind of delay to use, you can study the web access logs. The other way is to observe the users as they interact with the web site.

Timers are used to introduce a delay or pause before a sampler is run. In a test plan, even if a timer is placed after the samplers, it will run before the sampler. If the timer is a child element of a sampler, it will apply only to that sampler. Otherwise, it will apply to all the samplers in that scope. If multiple timers are in scope, all the timers will apply before a sampler is executed.

Constant Timer

The Constant Timer introduces a specified delay before the samplers in its scope are executed. The only configuration is the delay that is needed.

Let's look at the functionality with the help of an example. We will add a Constant Timer and configure it with a delay value of 5000 milliseconds or 5 seconds. We will make this a child element of a thread group so that it applies to all the samplers under the thread group.

Follow these steps or download ConstantTimerTestPlan.jmx.²⁸

1. Create a test plan and give it a meaningful name, such as Constant Timer Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /jmeter/alpha and **Method** as GET.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /jmeter;bravo and **Method** as GET.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /jmeter/zulu and **Method** as GET.
7. Click on Thread Group and go to Edit ▶ Add ▶ Timer. Add Constant Timer. Configure it with a value of 5000 milliseconds (see Figure 5-64).

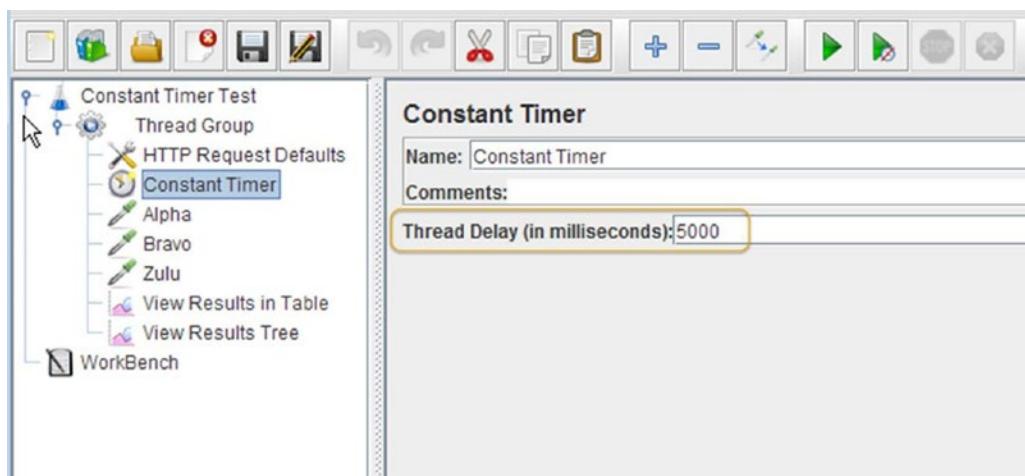


Figure 5-64. Constant Timer with delay of 5 seconds

²⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/ConstantTimerTestPlan.jmx

8. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
9. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
10. Save the test plan.
11. Run the test.

The results will be similar to those shown in Figure 5-65.

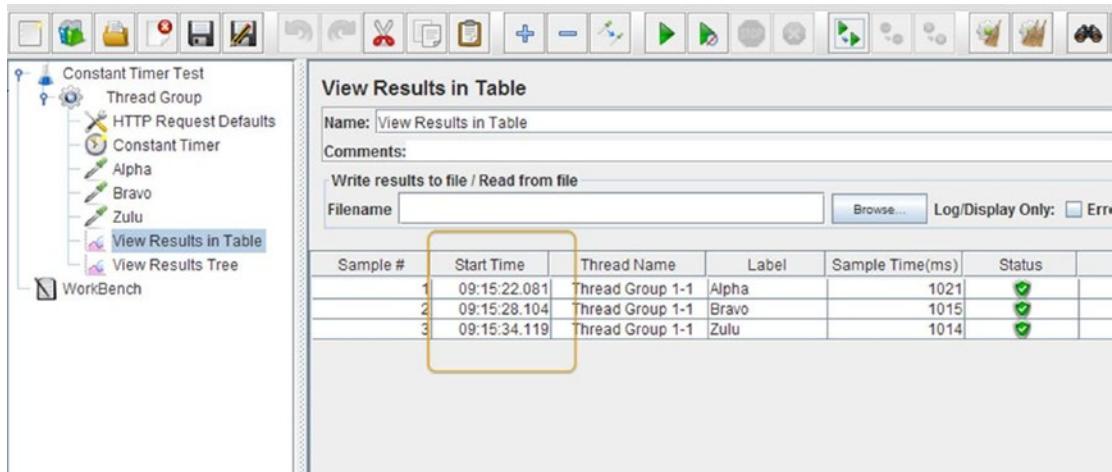


Figure 5-65. Constant Timer delayed 5 seconds

In these results, each of the HTTP requests has been delayed by five seconds.

As you saw in the previous example, the Constant Timer delays all the child elements in scope. What if you wanted to delay only a specific sampler by 5 seconds? You can achieve this by making the Constant Timer a child element of the sampler to be delayed.

Modify the example as follows:

1. Drag the Constant Timer and make it a child element of the HTTP request /jmeter/bravo. Leave the configuration as is (see Figure 5-66).

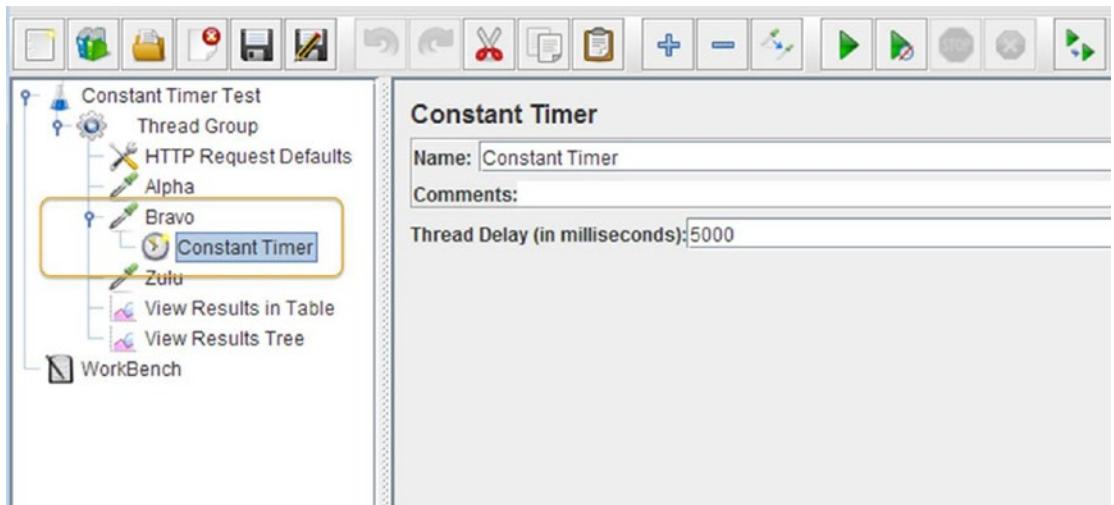


Figure 5-66. Constant Timer as child of bravo

2. Save the test plan.
3. Run the test.

The results will be similar to those shown in Figure 5-67.

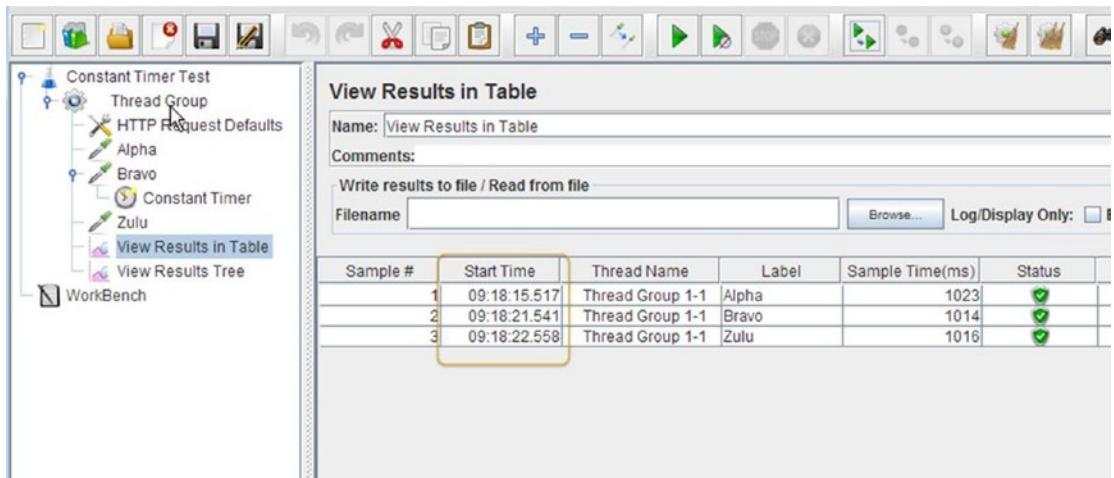


Figure 5-67. Constant Timer scoped by bravo

In these results, the HTTP request bravo has been delayed by 5 seconds, as it is in the scope of the Constant Timer. There is no delay before the HTTP request zulu.

Along with the other metrics, the Listeners display the time taken by the samplers. If you want to see the Timer delay included along with the sampler time, you use a Transaction Controller.

Follow these steps or download ConstantTimerAndTransactionCtrlTestPlan.jmx.²⁹

1. Open ConstantTimerTestPlan.jmx.
2. Go to File ▶ Save Test Plan as and type in a filename, such as ConstantTimerAndTransactionCtrlTestPlan.jmx.
3. Give the test plan a meaningful name, such as Constant Timer And Transaction Controller Test.
4. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Transaction Controller. Enable the **Generate Parent Sample** checkbox so that you will get only one entry for the Transaction Controller. Enable the **Include the Duration of the Timer and Pre-Post Processors in the Generated Sample** checkbox so that the Timer delay is included along with the HTTP request.
5. Drag the HTTP request /jmeter/bravo so it's the child element of the Transaction Controller (see Figure 5-68).

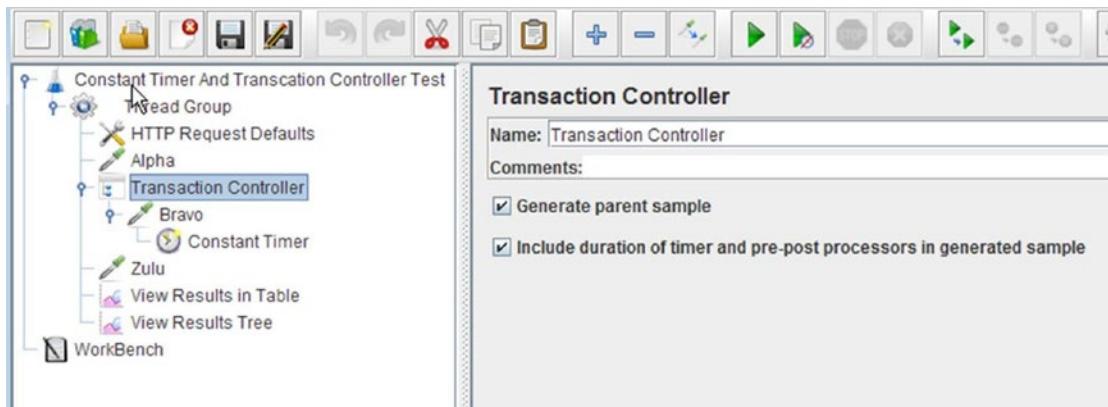


Figure 5-68. Constant Timer enclosing a HTTP request

6. Run the test.

The results will be similar to those shown in Figure 5-69.

²⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/ConstantTimerAndTransactionCtrlTestPlan.jmx

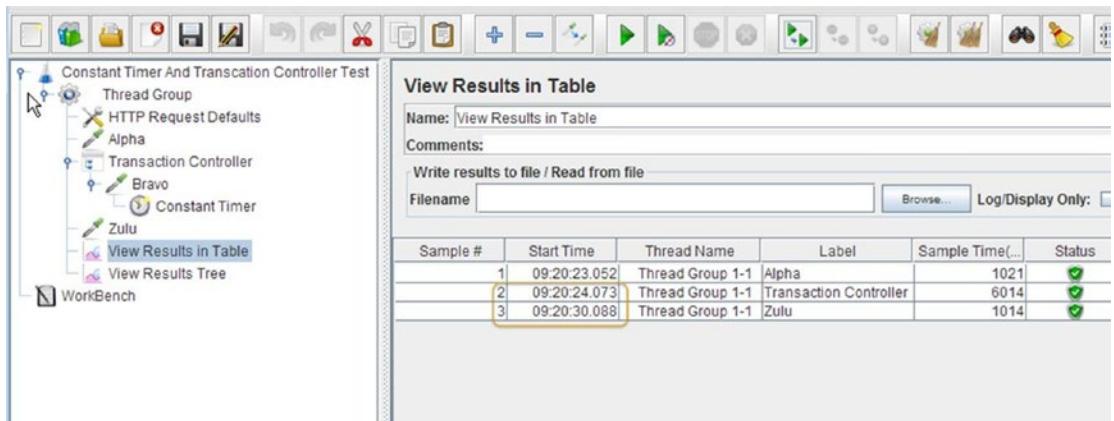


Figure 5-69. Constant Timer results

In these results, the Transaction Controller takes around six seconds, which includes five seconds for the Timer delay and one second for the /jmeter/bravo sampler.

Gaussian Random Timer

The Gaussian Random Timer introduces a delay according to the *Gaussian Distribution* (also called the bell curve). The delay varies around a central mean, as illustrated in Figure 5-70.

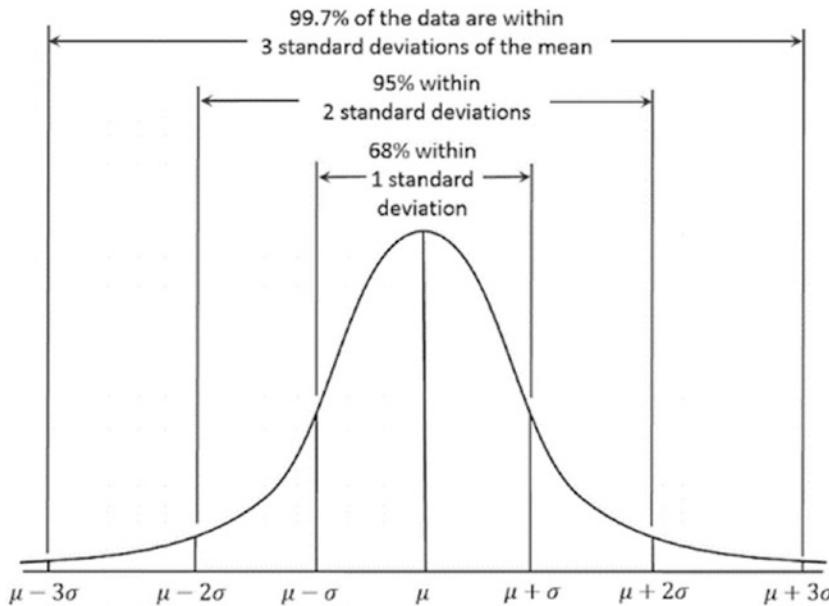


Figure 5-70. Gaussian Distribution or bell curve

Let's assume that the mean value is 10 seconds and the variance is 2 seconds. The delay introduced would vary as follows:

- 99% of the time, the delay would vary between $\text{mean} - \text{variance}$ and $\text{mean} + \text{variance}$ (between 4 seconds and 16 seconds).
- 95% of the time, the delay would vary between $\text{mean} - 2 * \text{variance}$ and $\text{mean} + 2 * \text{variance}$ (between 6 seconds and 14 seconds).
- 68% of the time, the delay would vary between $\text{mean} - 3 * \text{variance}$ and $\text{mean} + 3 * \text{variance}$ (between 8 seconds and 12 seconds).

Refer to Wikipedia for more details about the Gaussian Distribution.

Let's look at an example.

Follow these steps or download `GaussianTimerTestPlan.jmx`.³⁰

1. Create a test plan and give it a meaningful name, such as Gaussian Timer Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /jmeter/alpha and **Method** as GET.
5. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Loop Controller. Set the **Loop Count** to 10.
6. Click on Loop Controller and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /jmeter;bravo and **Method** as GET.
7. Click on the HTTP request /jmeter;bravo and go to Edit ▶ Add ▶ Timer. Add Gaussian Random Timer. Configure **Constant Delay Offset (in Milliseconds)** as 10000 and **Deviation (in Milliseconds)** as 2000 (see Figure 5-71).

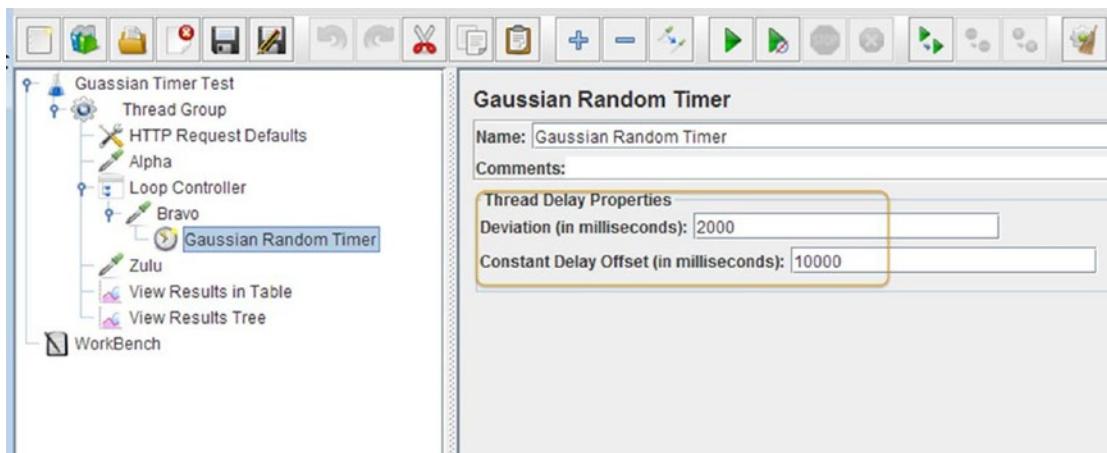


Figure 5-71. Gaussian Timer configuration

³⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/GaussianTimerTestPlan.jmx

8. Click on Thread Group and go to Edit > Add > Sampler. Configure **Path** as /jmeter/zulu and **Method** as GET.
9. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
10. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
11. Save the test plan.
12. Run the test.

The results will be similar to those shown in Figure 5-72.

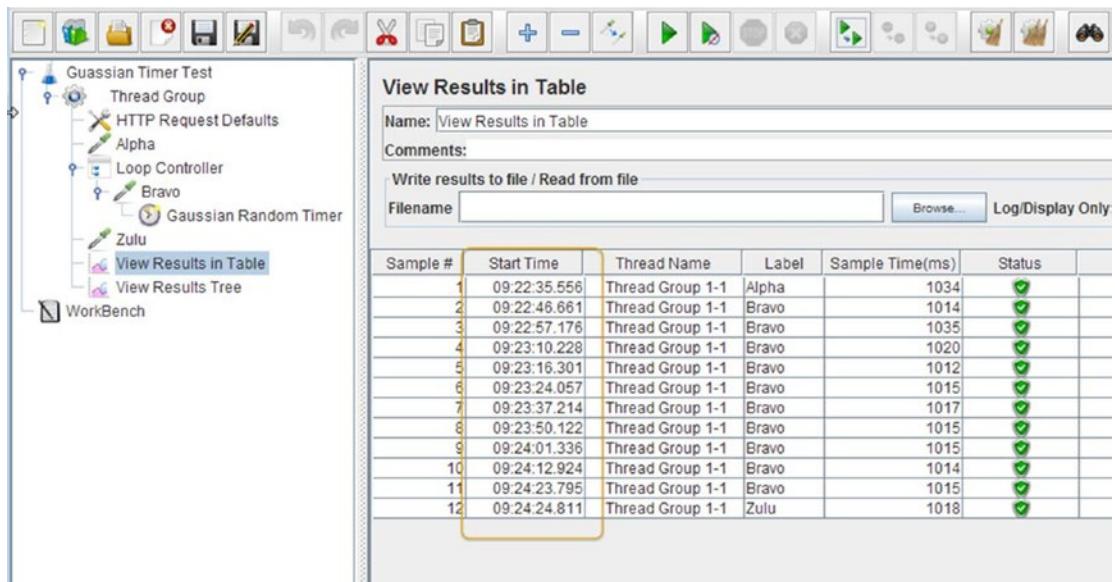


Figure 5-72. Gaussian Timer results

In these results, each of the requests to /jmeter/bravo has been delayed by a random delay centered about the mean value of 10 seconds.

Uniform Random Timer

The delay introduced by the Uniform Random Timer has two parts:

- **Constant Delay:** Fixed and equal to the configured value.
- **Random Delay:** Varies between zero and the configured value.

The actual delay will range between the Constant Delay and the Constant Delay plus the Random Delay. As the term “uniform” indicates, the delay varies within its range with equal probability.

If the Constant Delay was configured to 10 seconds and the Random Delay was configured to 5 seconds, then the actual delay will vary between 10 and 15 seconds.

Let's look at the following example.

Follow these steps or download `UniformRandomTimerTestPlan.jmx`.³¹

1. Create a test plan and give it a meaningful name, such as **Uniform Random Timer Test**.
2. Click on Test Plan and go to Edit > Add > Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit > Add > Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit > Add > Sampler. Configure **Path** as /jmeter/alpha and **Method** as GET.
5. Click on Thread Group and go to Edit > Add > Logic Controller. Add Loop Controller. Set the **Loop Count** to 10.
6. Click on Loop Controller and go to Edit > Add > Sampler. Add HTTP Request. Configure **Path** as /jmeter;bravo and **Method** as GET.
7. Click on HTTP request /jmeter;bravo and go to Edit > Add > Timer. Add Uniform Random Timer. Configure **Random Delay Maximum (in Milliseconds)** as 5000 and **Constant Delay Offset (in Milliseconds)** as 10000 (see Figure 5-73).

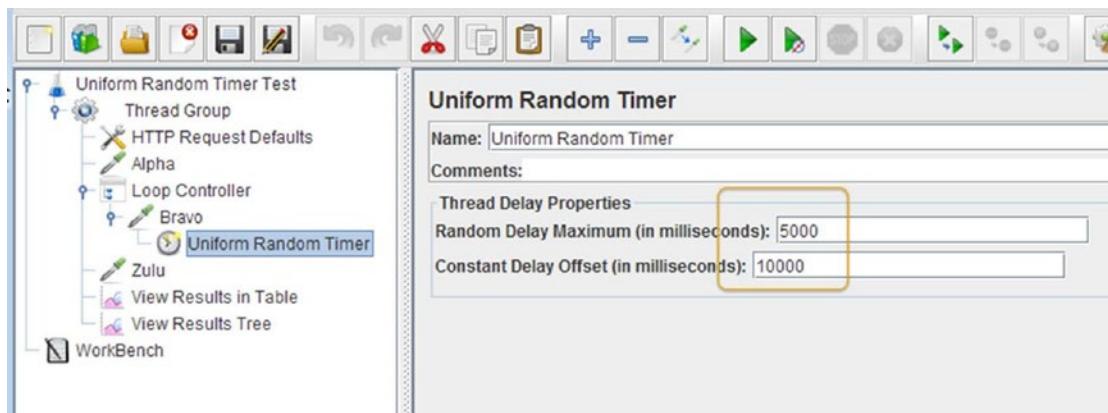


Figure 5-73. Uniform Random Timer configuration

8. Click on Thread Group and go to Edit > Add > Sampler. Configure **Path** as /jmeter/zulu and **Method** as GET.
9. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
10. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.

³¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/UniformRandomTimerTestPlan.jmx

11. Save the test plan.
12. Run the test.

The results will be similar to those shown in Figure 5-74.

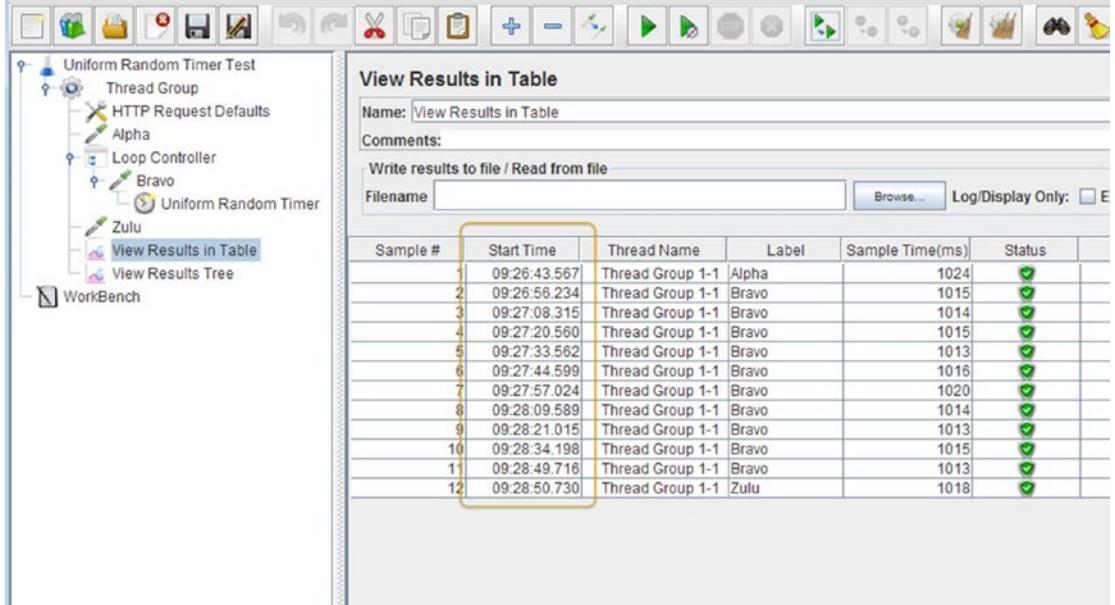


Figure 5-74. Uniform Random Timer results

In these results, each of the requests to /jmeter/bravo has been delayed by a random delay varying between 10 and 15 seconds.

Constant Throughput Timer

The Constant Throughput Timer calculates and introduces delays between samplers so as to keep the throughput at the configured value.

Let's illustrate this with an example.

Follow these steps or download `ConstantThroughputTimerTestPlan.jmx`.³²

1. Create a test plan and give it a meaningful name, such as **Constant Throughput Timer Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group. Configure **Number of Threads (Users)** as 1 and **Loop Count** as 12.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to **Edit > Add > Sampler**. Configure **Path** as /jmeter/alpha and **Method** as GET.

³²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/ConstantThroughputTimerTestPlan.jmx

- Click on Thread Group and go to Edit ▶ Add ▶ Timer. Add Constant Throughput Timer. Configure **Target Throughput (in Samples per Minute)** as 4.0 and **Calculate Throughput Based on** as All Active Threads (Shared) (see Figure 5-75).

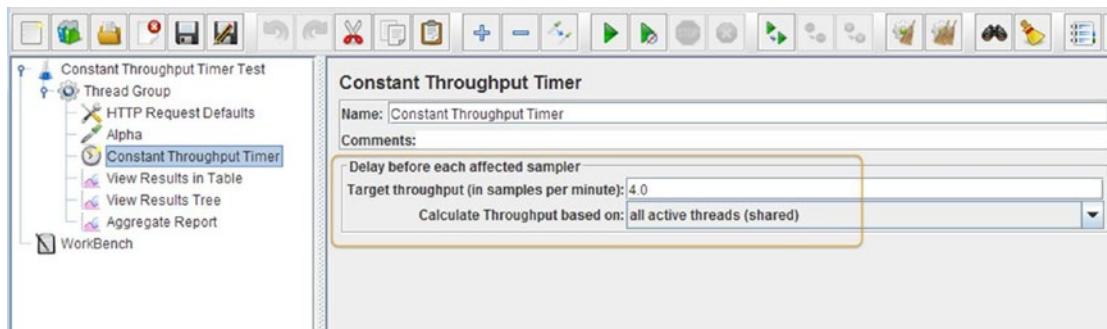


Figure 5-75. Constant Throughput Timer set to four requests per minute

- Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
- Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
- Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add Aggregate Report.
- Save the test plan.
- Run the test.

The results will be similar to those shown in Figure 5-76.

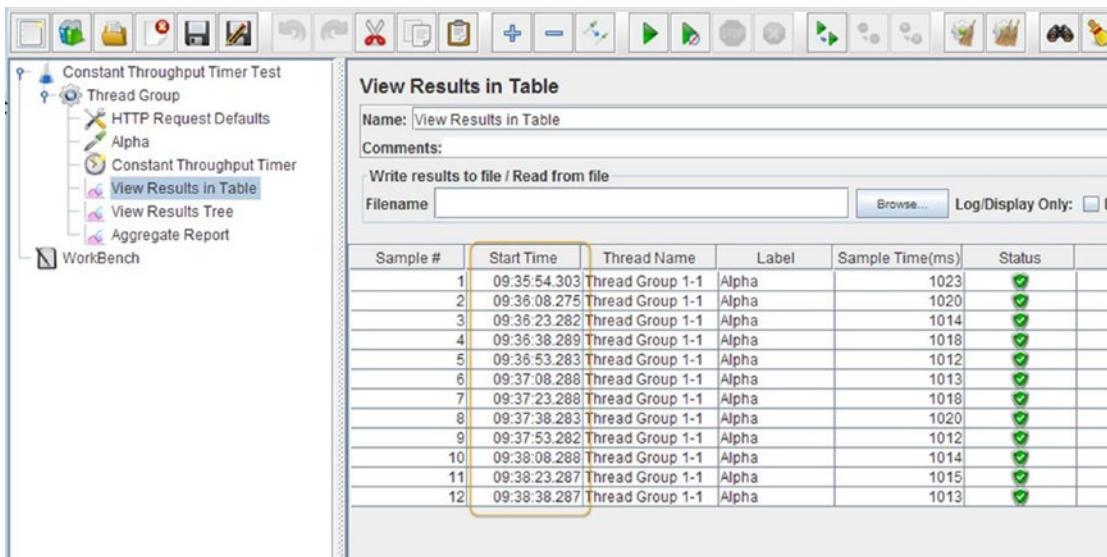


Figure 5-76. Constant Throughput Timer results

In these results, each of the requests to /jmeter/alpha has been delayed by a varying amount so as to keep the overall throughput at the configured level (see Figure 5-77).

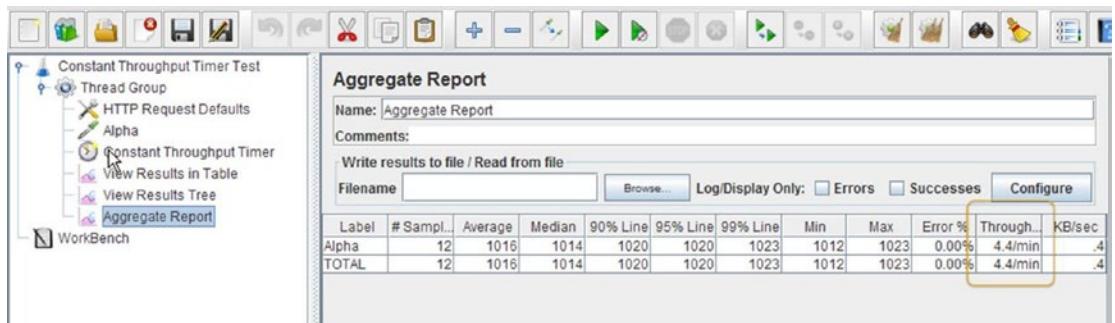


Figure 5-77. Constant Throughput Timer configure aggregate

In the Aggregate results, observe that the throughput is 4.4 requests per minute, which is close enough to the configured value of 4 requests per minute.

Synchronizing Timer

The Synchronizing Timer blocks threads and releases them all at once, thus creating a large load at the same instant. This is very helpful to test how the application handles simultaneous requests.

Let's illustrate this with an example.

Follow these steps or download SynchronizingTimerTestPlan.jmx.³³

1. Create a test plan and give it a meaningful name, such as Constant Throughput Timer Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (Users)** as 4, **Loop Count** as 12, and **Ramp-Up** as 0.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Configure **Path** as /jmeter/alpha and **Method** as GET.
5. Click on Thread Group and go to Edit ▶ Add ▶ Timer. Add Synchronizing Timer. Configure **Number of Simulated Users to Group by** as 2 and **Timeout in Milliseconds** as 3000 (see Figure 5-78).

³³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/timers/SynchronizingTimerTestPlan.jmx

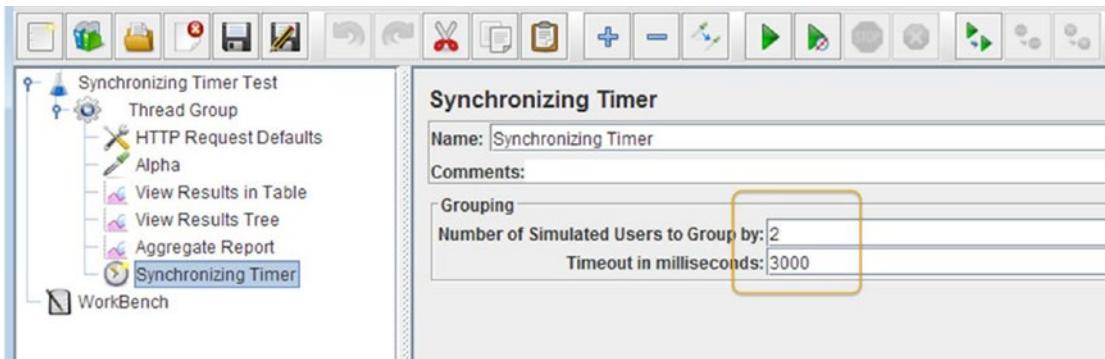


Figure 5-78. Synchronizing Timer configuration

6. Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree.
7. Click on Thread Group and go to Edit > Add > Listener. Add View Results in Table.
8. Click on Thread Group and go to Edit > Add > Listener. Add Aggregate Report.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-79.

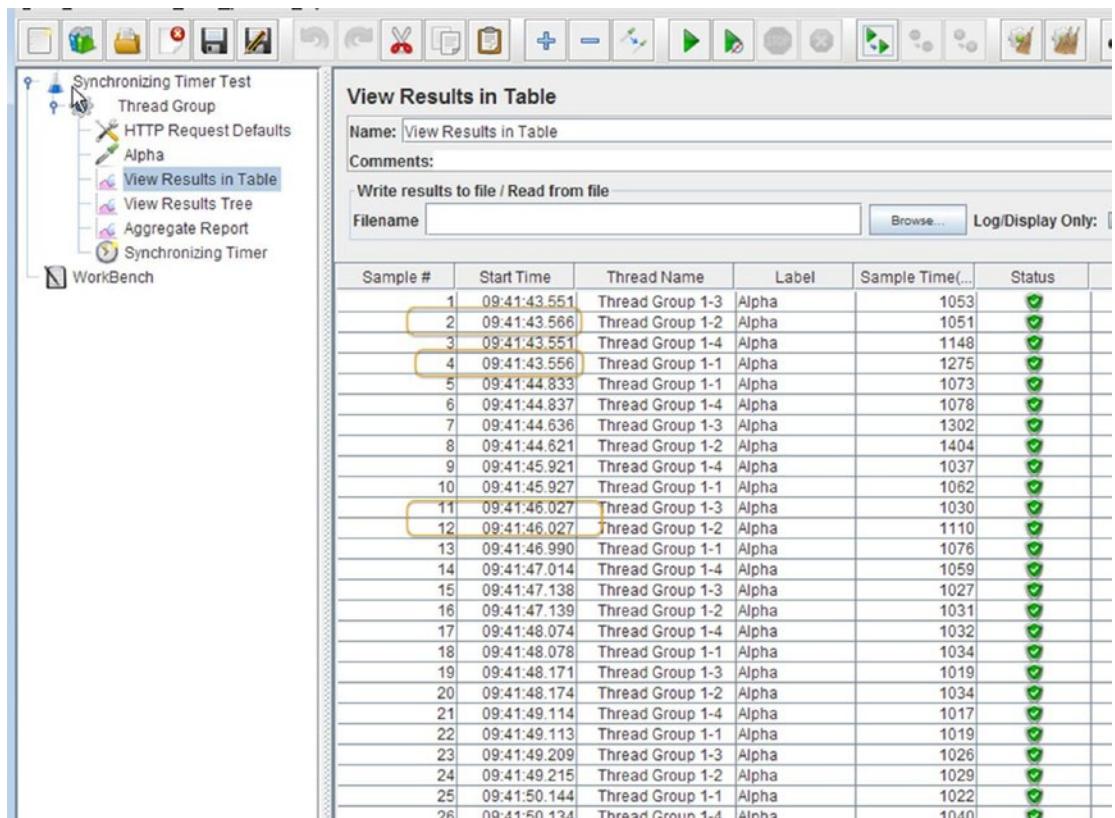


Figure 5-79. Synchronizing Timer results

In these results, each of the requests to /jmeter/alpha has been delayed so that both the samples are released at the same time, as marked in red.

Sampler

The Sampler is a component that's used to send requests to the application being tested. If the test plan has more than one sampler, they will be executed in the order they are defined in the test plan tree.

JMeter has the following samplers as of release 3.0:

- *Access Log Sampler*
- *AJP/1.3 Sampler*
- *BeanShell Sampler*
- *BSF Sampler*
- *Debug Sampler*
- *FTP Request*
- *HTTP Request*

- *Java Request*
- *JDBC Request*
- *JMS Point-to-Point*
- *JMS Publisher*
- *JMS Subscriber*
- *JSR223 Sampler*
- *JUnit Request*
- *LDAP Extended Request*
- *LDAP Request*
- *Mail Reader Sampler*
- *OS Process Sampler*
- *SMTP Sampler*
- *SOAP/XML-RPC Request*
- *TCP Sampler*
- *Test Action*

Sampler properties can be configured as per requirements. However, most of the time, the default configuration is sufficient.

Each of the samplers generates one or more sample results (the exception is Test Action). These sampler results have various attributes, such as success/fail/elapsed time, etc., which are viewed by using various listeners.

With each sampler, you should use assertions to make sure that the sampler is working as per your requirements. For example, you would use response assertions for the HTTP request to check for status 200.

HTTP Request

The *HTTP Request* sampler is used when you want to use POST, GET, DELETE, PUT, etc., methods over HTTP(S) on the target web application (*Sampler AJP/1.3 sampler* also has these methods).

The minimum configuration required for a HTTP Request test element to work is the server name or IP. If the port number is 80, it can be omitted. You can specify timeouts (in milliseconds) while waiting for a response from the server.

JMeter provides several properties to configure the HTTP Request sampler. Let's walk through each of these with examples.

Implementing the HTTP Request Sampler

The configuration field Implementation can have any of the following values:

- **HttpClient4:** This uses Apache HttpClient version 4 component.
- **HttpClient3.1:** This uses Apache HttpClient version 3.1 component.
- **Java:** This uses the HTTP implementation provided by the JVM.
- **Blank value:** This uses a value defined in the HTTP Request Defaults configuration element.

If it's not defined, it relies on the `jmeter.httpSampler` property defined in the `jmeter.properties` file, failing which it defaults to `HttpClient4`.

`HttpClient` internally uses Apache HTTP components.³⁴

A request with a blank value or `HttpClient` will send a HTTP header for `UserAgent` with a value of `Apache-HttpClient/4.2.6 (java 1.5)`. However, if the implementation type is Java, then the HTTP header for `UserAgent` will not be sent.

Let's look at the following example.

Follow these steps or download `HTTPClientImplTestPlan.jmx`.³⁵

1. Create a test plan and give it a meaningful name, such as `HTTP Client Implementation Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Config Element`. Add `HTTP Cookie Manager`.
4. Click on Thread Group and go to `Edit > Add > Sampler`. Add `HTTP Request`. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, **Path** as `/user/signIn`, and **Method** as `POST`. Choose **Implementation as Java** from the drop-down. Add the parameters, **Name/Value** as `email/user1@dt.com` and `password/user1`.
5. Click on `HTTP Request` and go to `Edit > Add > Assertions`. Add `Response Assertion`. Configure **Response Field to Test** as `Response Code`, **Pattern Matching Rules** as `Equals`, and **Patterns To Test** as `200`.
6. Click on Thread Group and go to `Edit > Add > Sampler`. Add `HTTP Request`. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, **Path** as `/user/signOut`, and **Method** as `HEAD`. Leave the **Implementation** as blank.
7. Click on `HTTP Request` and go to `Edit > Add > Assertions`. Add `Response Assertion`. Configure **Response Field to Test** as `Response Code`, **Pattern Matching Rules** as `Equals`, and **Patterns To Test** as `200`.
8. Click on Thread Group and go to `Edit > Add > Listener`. Add `View Results Tree` (see Figure 5-80).

³⁴<https://hc.apache.org/httpcomponents-client-ga>

³⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/HTTPClientImplTestPlan.jmx

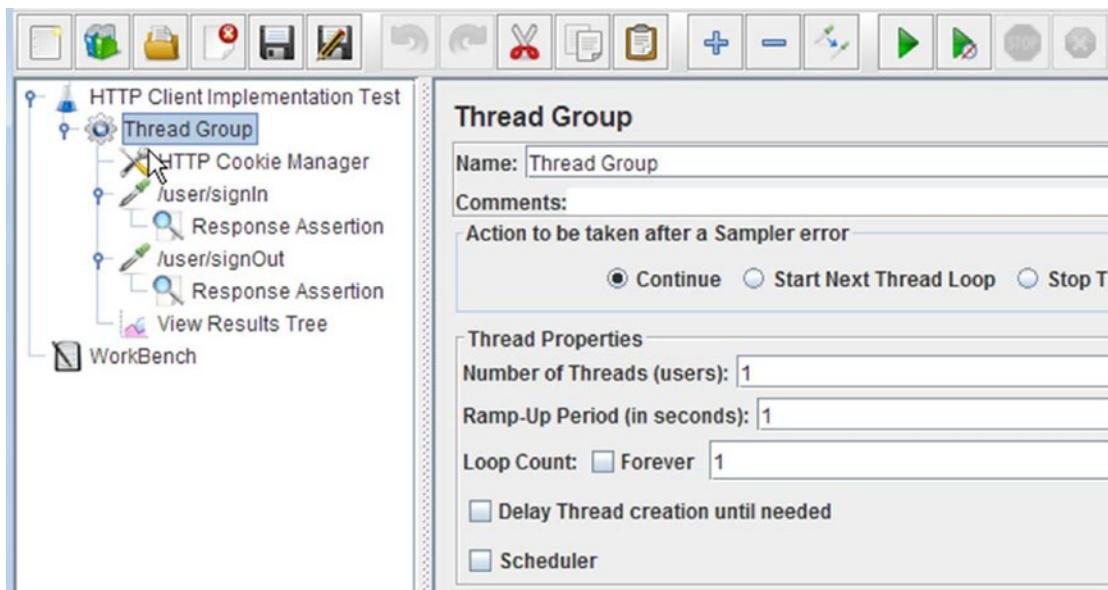


Figure 5-80. HTTP client implementation test

9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-81.

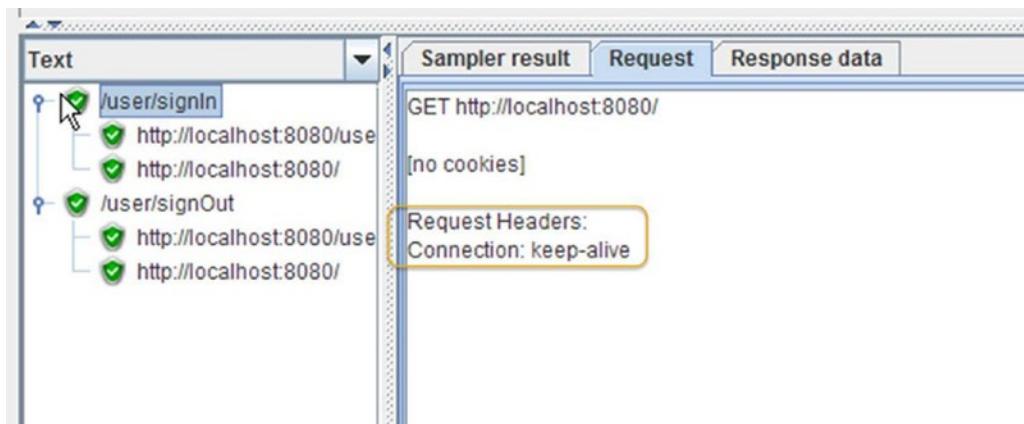


Figure 5-81. HTTP Request HTTPClient as Java

In these results, you will not see the User-Agent header in the first HTTP request, due to the Implementation option being set to Java (see Figure 5-82).

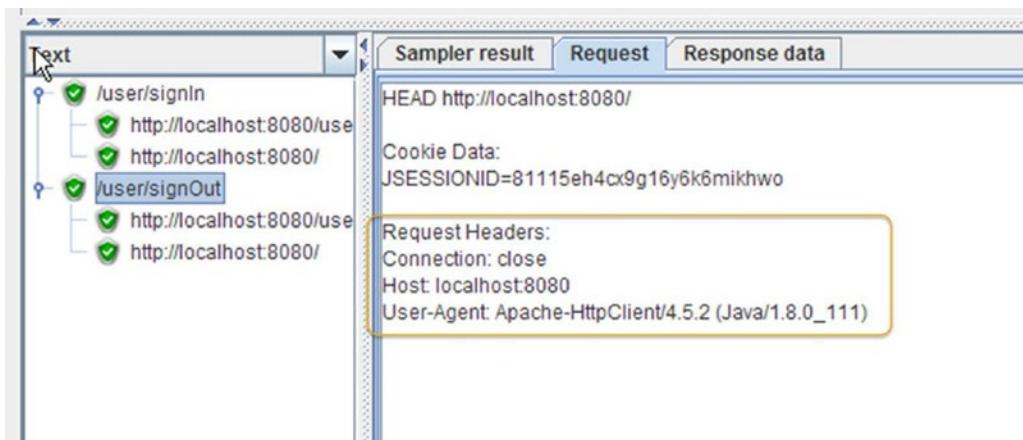


Figure 5-82. HTTP Request HTTPClient as a blank value

Protocols

JMeter provides `http`, `https`, and `file` protocols. We have already walked through the `http` protocol. Now, let's look at the procedure to record the tests on web sites that use the `https` protocol.

From JMeter 2.11 onwards, a dummy certificate is created in the `/bin` directory as soon as you start the proxy server from the `HTTP(S) Test Script Recorder`. It creates two files, which you can see under the `/bin` directory.

```
C:\apache-jmeter-3.0>cd bin
C:\apache-jmeter-3.0\bin>Dir ApacheJMeter*
Volume in drive C has no label.
Volume Serial Number is DA32-01EE

Directory of C:\apache-jmeter-3.0\bin

05/13/2016  11:22 PM           12,885 ApacheJMeter.jar
04/22/2017  03:11 AM           1,328 ApacheJMeterTemporaryRootCA.crt
                           2 File(s)      14,213 bytes
                           0 Dir(s)    7,217,209,344 bytes free
```

C:\apache-jmeter-3.0\bin>

We need to import `ApacheJMeterTemporaryRootCA.crt` into the browser being used for recording. Let's see this with the help of the following example, using the Firefox browser.

1. Open JMeter GUI.
2. Click on WorkBench and go to `Edit > Add > Non-Test Elements`. Add `HTTP(S) Test Script Recorder`. Configure the Target Controller by selecting `WorkBench > HTTP(S) Test Script Recorder`.
3. Launch the Firefox browser and go to `Preferences > Advanced > Certificates > View Certificates`. Click on `Import`, and then select the `ApacheJMeterTemporaryRootCA.crt` certificate (see Figure 5-83).

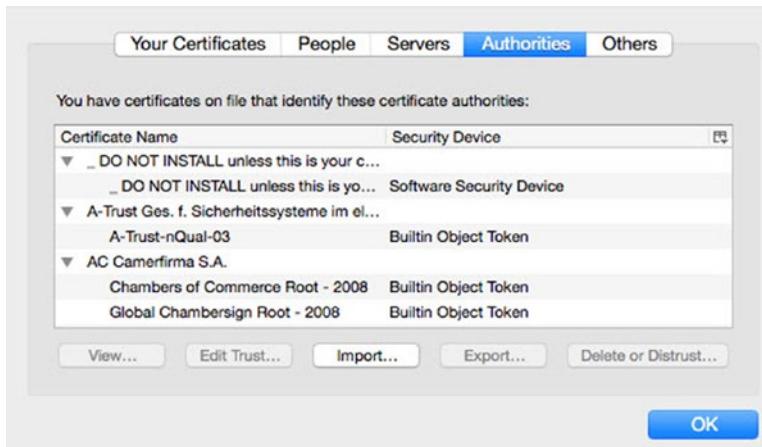


Figure 5-83. Firefox certificate

4. Open the Firefox browser and launch <https://in.yahoo.com>.
5. Stop recording.
6. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
7. Move all the recorded samplers as child elements of the thread group.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
9. Run the test and you will see that the https requests are getting executed.

Note The file protocol is only for testing.

Redirect Automatically

The Redirect Automatically configuration field sets the underlying http protocol handler to automatically follow redirects. They are not seen by JMeter and thus will not appear as samples. This will be helpful when you don't want to see the requests that are redirected. This should only be used for GET and HEAD requests, as the HttpClient sampler will reject attempts to use it for POST or PUT.

You can use either the Redirect Automatically or Follow Redirects configuration property, but not both at the same time.

Let's illustrate this by the following example.

Follow these steps or download [RedirectAutoTestPlan.jmx](#).³⁶

1. Create a test plan and give it a meaningful name, such as Redirect Automatically Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.

³⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/RedirectAutoTestPlan.jmx

3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signIn, and **Method** as POST. Add the parameters, **Name/ Value** as email/user1@dt.com and password/user1.
5. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signOut, and **Method** as HEAD. Enable the **Redirect Automatically** checkbox (see Figure 5-84).

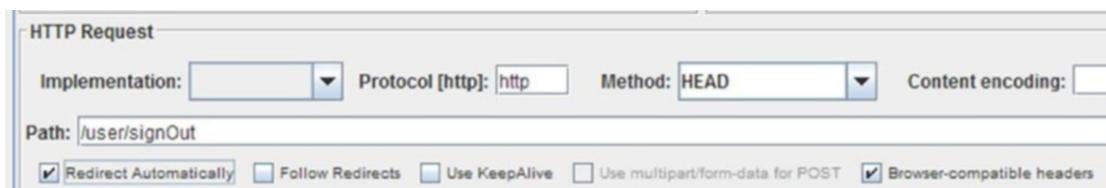


Figure 5-84. Redirect Automatically checkbox

7. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200 (see Figure 5-85).

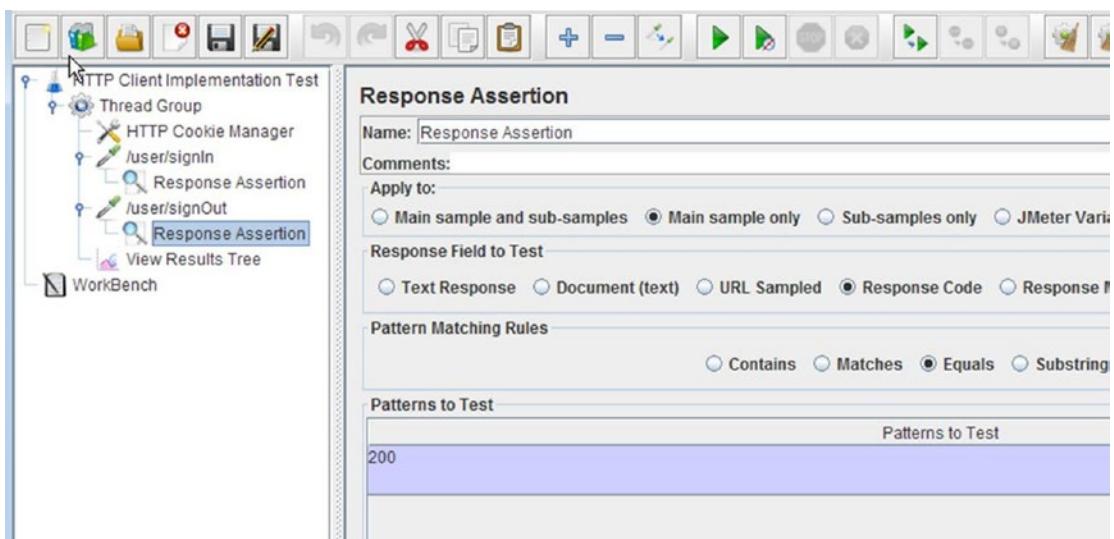


Figure 5-85. Redirect Automatically test

8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
9. Save the test plan.
10. Run the test.

The results will be similar to those shown in Figure 5-86.

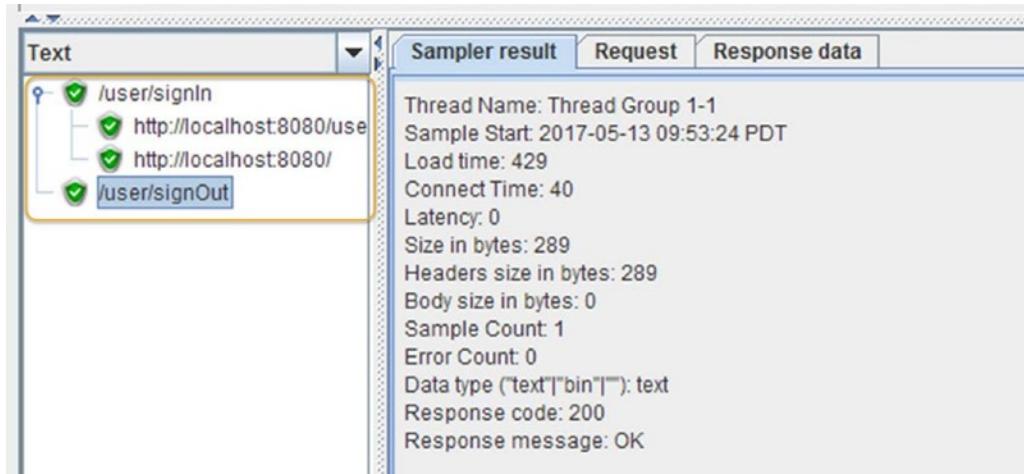


Figure 5-86. View Results Tree Redirect Automatically

Follow Redirect

On the web, the URLs for the web pages or resources change frequently. The browser is informed of the new URL by the concept of URL redirection. This is also referred to as URL forwarding. HTTP status codes 3XX are used to indicate redirection.

Let's illustrate this with an example.

Follow these steps or download [FollowRedirectTestPlan.jmx](#).³⁷

1. Create a test plan and give it a meaningful name, such as Follow Redirect Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signIn, and **Method** as POST. Add the parameters, **Name / Value** as email/user1@dt.com and password/user1.
5. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.

³⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/FollowRedirectTestPlan.jmx

- Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signOut, and **Method** as HEAD. By default, the **Follow Redirects** flag is enabled (see Figure 5-87).

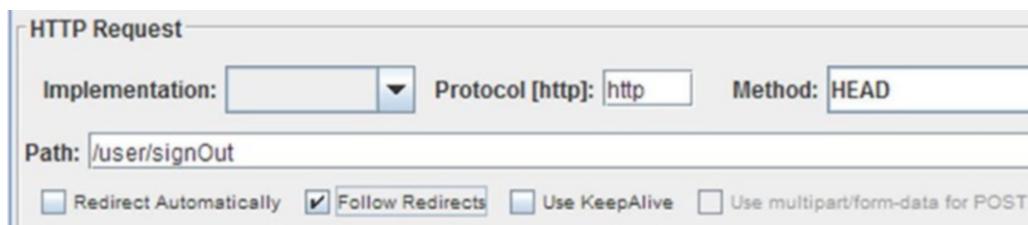


Figure 5-87. Using Follow Redirect

- Click on HTTP Request and go to Edit > Add > Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
- Click on Thread Group and go to Edit > Add > Listener. Add View Results Tree (see Figure 5-88).

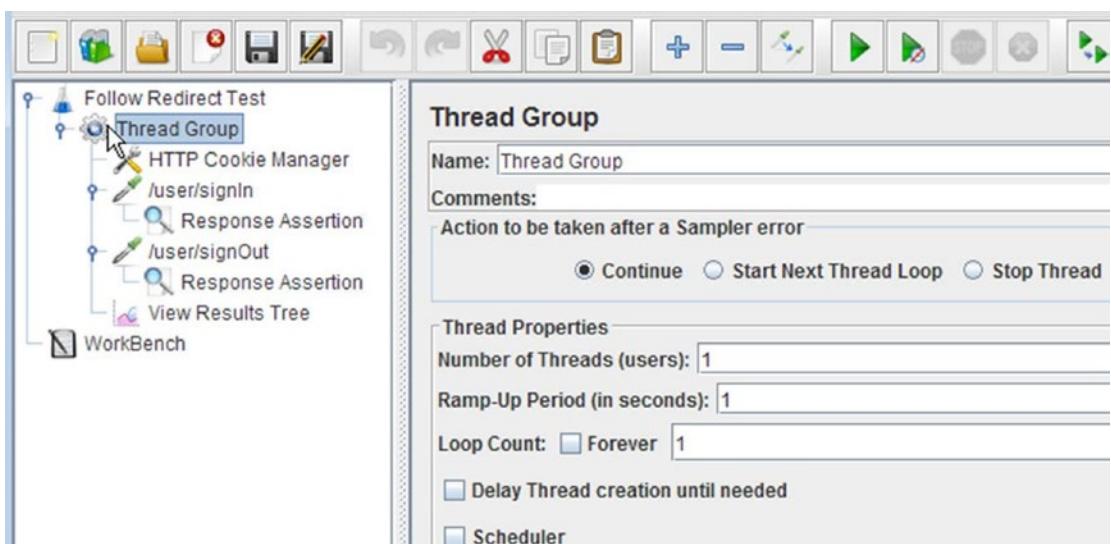


Figure 5-88. Follow Redirect test

- Save the test plan.
- Run the test.

The results will be similar to those shown in Figure 5-89.

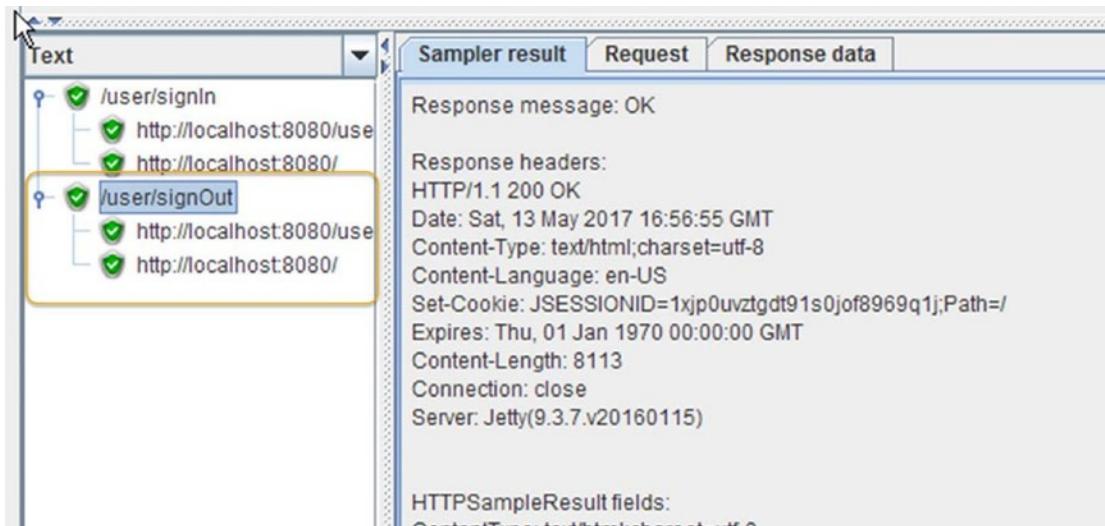


Figure 5-89. View Results Tree Follow Redirect

11. Remove the selection of Follow Redirects flag in the HTTP Request sampler.
12. Run the test.

The results will be similar to those shown in Figure 5-90.

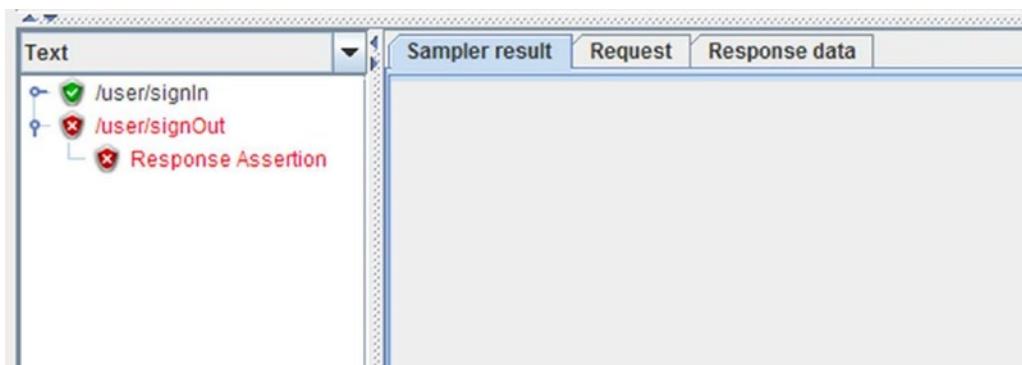


Figure 5-90. Test failure

In this case, assertions has been configured to match the Response Code of 200. But the redirected response code received was 302, so the test failed.

Use KeepAlive

The Use KeepAlive checkbox is enabled by default. This allows the same connection to be re-used by including KeepAlive in the Connection header.

Let's illustrate this by the following example.

Follow these steps or download [KeepAliveTestPlan.jmx](#).³⁸

1. Create a test plan and give it a meaningful name, such as Keep Alive Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signIn, and **Method** as POST. Add the parameters, **Name/ Value** as email/user1@dt.com and password/user1.
5. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signOut, and **Method** as HEAD. Disable the **Use KeepAlive** checkbox.
7. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 5-91).

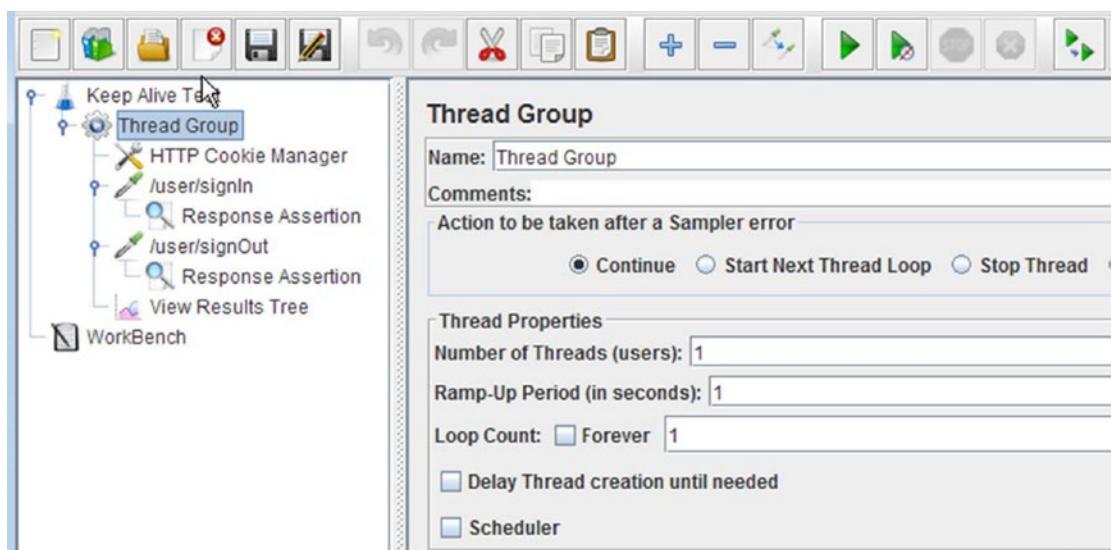


Figure 5-91. Keep Alive test

³⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/KeepAliveTestPlan.jmx

9. Save the test plan.

10. Run the test.

The results will be similar to those shown in Figure 5-92.

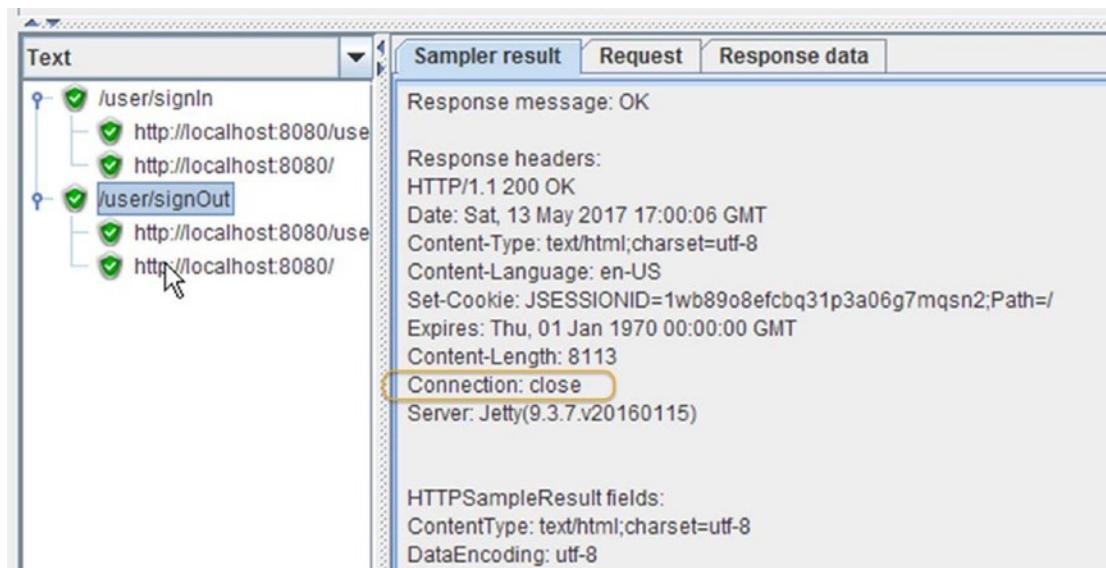


Figure 5-92. View Results Tree UseKeepAlive

Note If you disable the KeepAlive configuration option for the last request, then the next request will use a new connection.

Use Multipart/Form-Data for POST

When you make a POST request, you have to encode the data that forms the body of the request in some format. When this property is enabled, it indicates that this request contains file data and allows entire files to be included in the data. The file needs to be supplied through the Send Files With the Request input.

If this property is not enabled, it will use application/x-www-form-urlencoded format.

Browser-Compatible Headers

This property is used in combination with multipart/form-data for POST. Using this suppresses the Content-Type and Content-Transfer-Encoding Headers. Only the Content-Disposition Header is sent.

When it's not checked, it will use Content-Transfer-Encoding: binary.

Send Parameter with the Request

With this property, you can specify the request parameters as name/value pairs. Enabling the Encode? checkbox encodes the special characters in the name/value pair. It is always best to enable this. Enabling the Include Equals? checkbox will force an '=' character even if the value is empty.

The Send Parameters With the Request option has a Detail button that is useful while reviewing/modifying verbose values (see Figure 5-93).

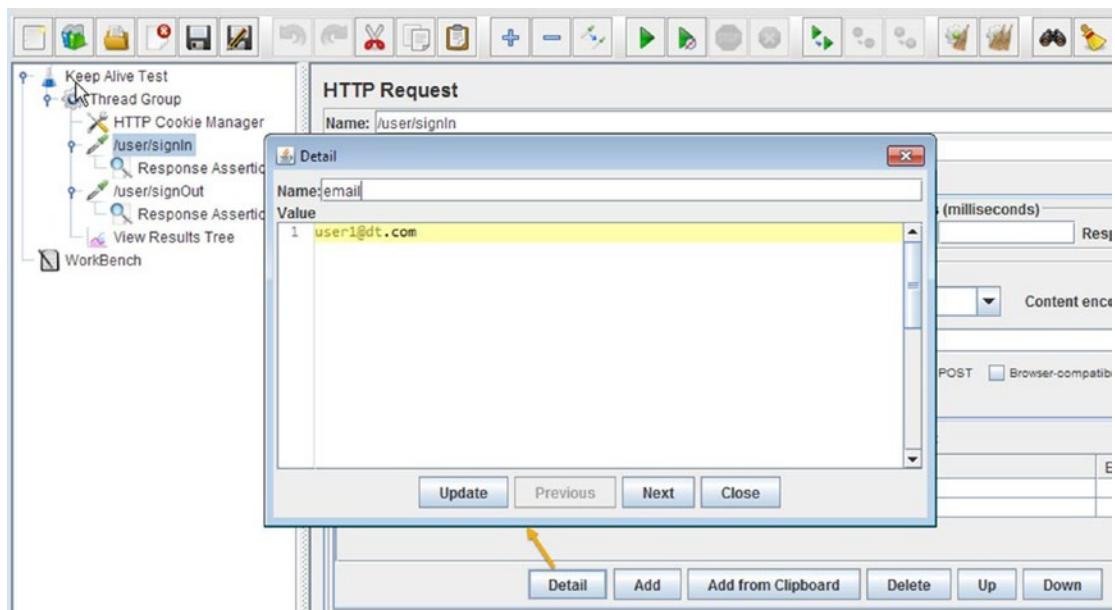


Figure 5-93. Detail button

Body Data

For Body Data, you need to prepare an URL if it has special characters.

URL encoding starts with % and then includes the equivalent Hexadecimal code; refer to [asciitable³⁹](http://www.asciiitable.com) to determine the other special characters in an URL.

Let's illustrate this by the following example.

Follow these steps or download [BodyDataTestPlan.jmx](#).⁴⁰

1. Create a test plan and give it a meaningful name, such as Body Data Test.
2. Click on Test Plan and go to Edit > Add > Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit > Add > Config Element. Add HTTP Cookie Manager.

³⁹<http://www.asciiitable.com>

⁴⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/BodyDataTestPlan.jmx

- Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signIn, and **Method** as POST. Click on **Body Data** and add the post parameters as email=user1%40dt.com&password=user1 (see Figure 5-94).

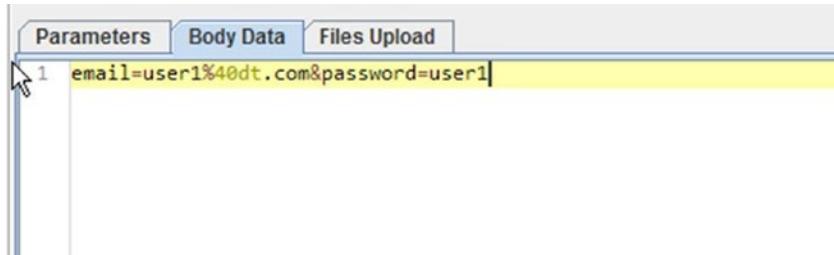


Figure 5-94. Send body data over POST request

- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
- Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /user/signOut.
- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
- Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 5-95).

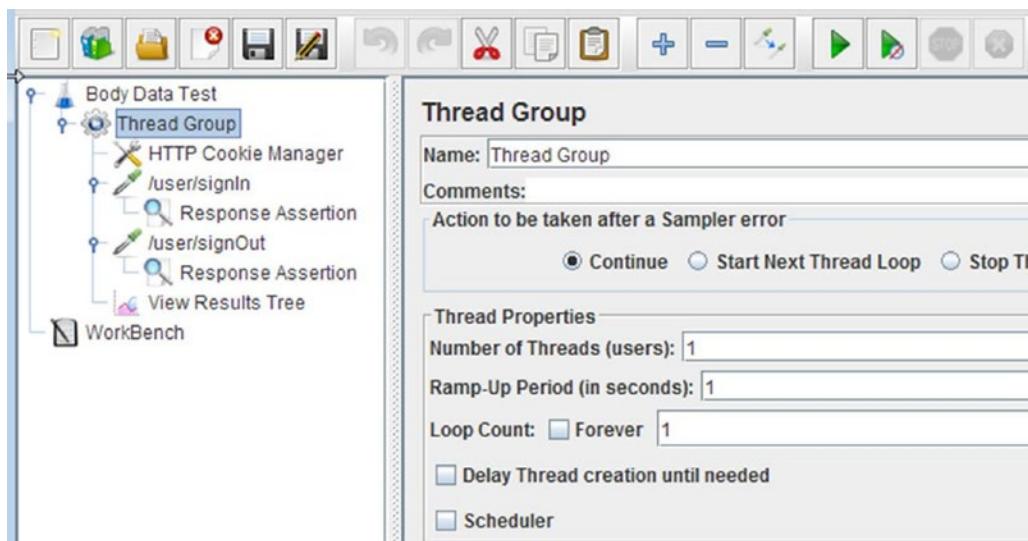


Figure 5-95. Body Data test

9. Save the test plan.

10. Run the test.

The results will be similar to those shown in Figure 5-96.

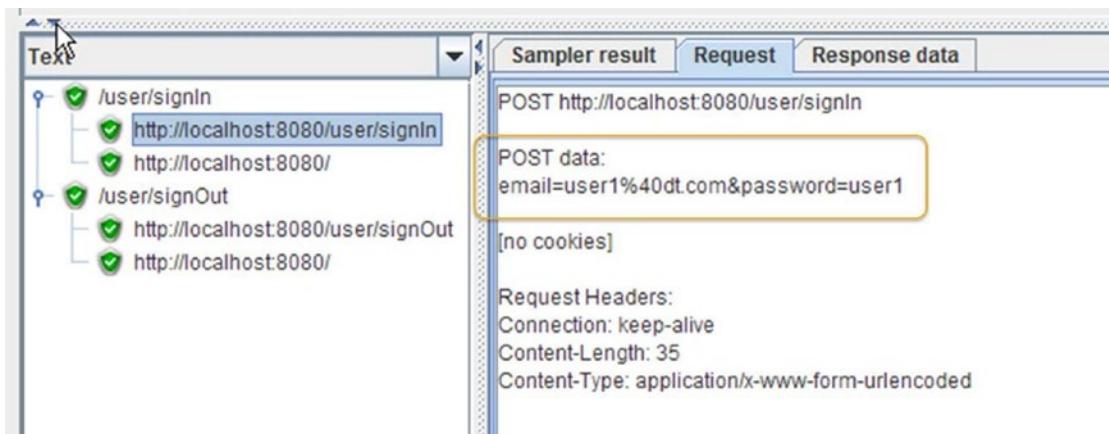


Figure 5-96. View Results Tree Body Data

This option is also useful with the following:

- GWT RPC HTTP Request
- JSON REST HTTP Request
- XML REST HTTP Request
- SOAP HTTP Request

Each line in the Body Data field, except the last line, is sent with CRLF appended. To send a CRLF after the last line of data, just ensure that there is an empty line following it. (This cannot be seen, except by noting whether the cursor can be placed on the subsequent line.)

Switching Between Name:Value and Body Data

Switching between the Name:Value and Body Data fields requires you to clear the present set of data. For example, if you are in Body Data and want to move to Name:Value, you have to wipe out all the Name:Value pairs before moving to Body Data, and vice versa (see Figure 5-97).



Figure 5-97. Switching between Name:Value and Body Data

Send Files with the Request

This property is used to send files over HTTP via a POST/PUT/PATCH request. You have to provide a file path, a parameter name, and the MIME type.

The file path is either the absolute path or the path where JMeter starts (that is, the root directory where JMeter starts). You can see this in the following example.

Follow these steps or download `FileLoadTestPlan.jmx`.⁴¹

1. Create a test plan and give it a meaningful name, such as `File Load Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Config Element`. Add HTTP Cookie Manager.
4. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, **Path** as `/dt/fileUpload`, and **Method** as `GET`.
5. Click on HTTP Request and go to `Edit > Add > Assertions`. Add Response Assertion. Configure **Response Field to Test** as `Response Code`, **Pattern Matching Rules** as `Equals`, and **Patterns To Test** as `200`.
6. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, **Path** as `/dt/fileUpload`, and **Method** as `POST`. Under **Send Files With the Request**, add `fileload.txt`. Select the checkbox for **Use Multipart/ Form-Data for POST and Browser-Compatible Headers** (this is required while uploading files with a POST request) (see Figure 5-98).

⁴¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/FileLoadTestPlan.jmx

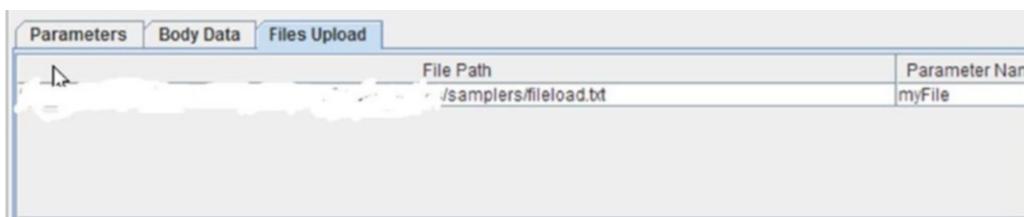


Figure 5-98. File load parameter name

7. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
8. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Document (text), **Pattern Matching Rules** as Contains, and add **Pattern To Test** as Success!. Add another pattern as File uploaded successfully.
9. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 5-99).

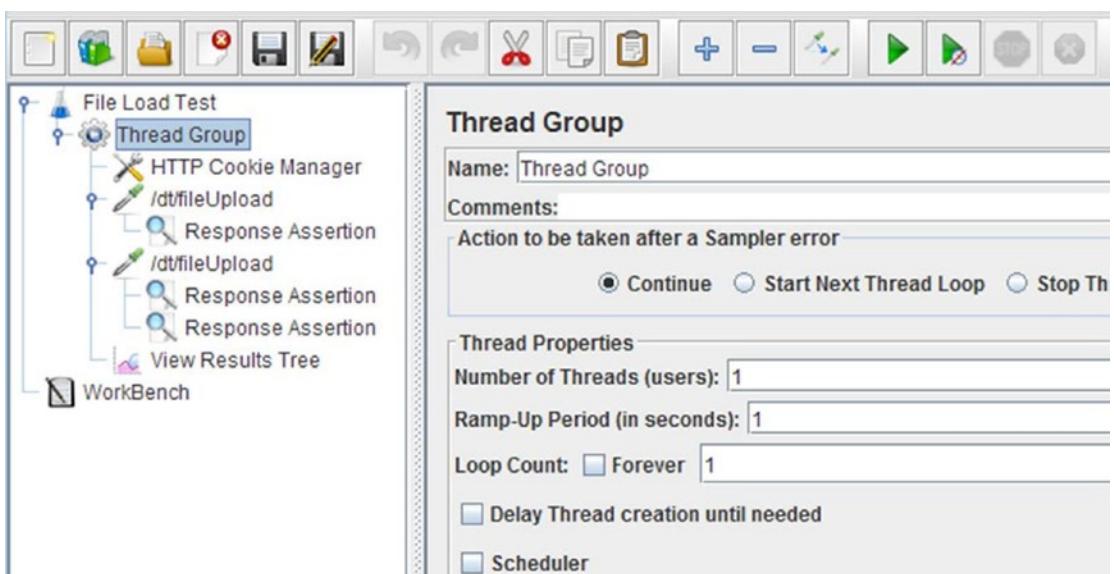


Figure 5-99. File Load test

10. Save the test plan.

11. Run the test.

The results will be similar to those shown in Figure 5-100.

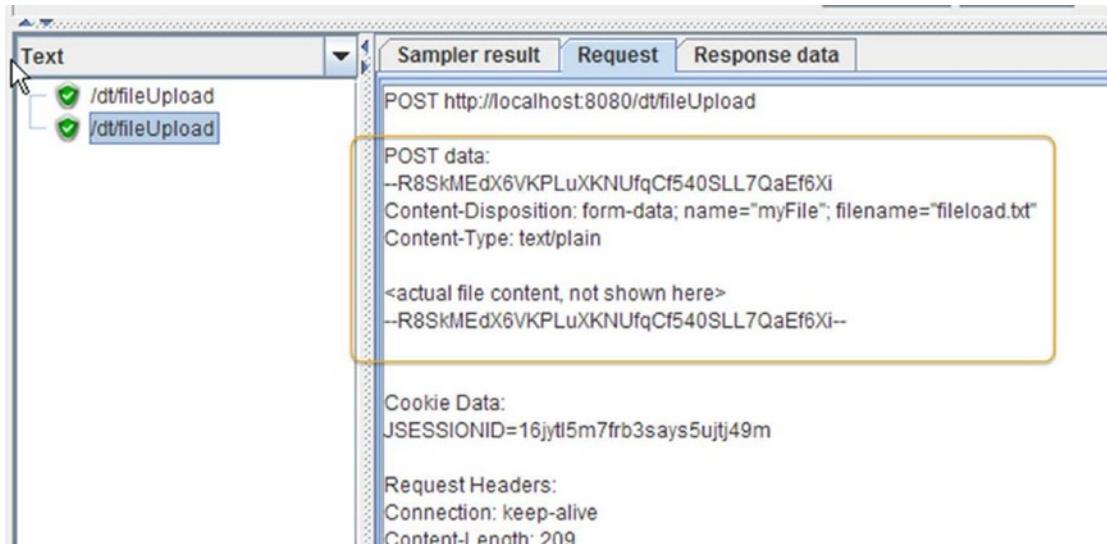


Figure 5-100. View Results Tree file load

The MIME type is the content type of the file being used in the request. It is good practice to provide the MIME type, as the web server may behave based on the MIME type and on the application being tested.

The parameter name used in the HTTP request is taken from the HTML code (see Figure 5-101).

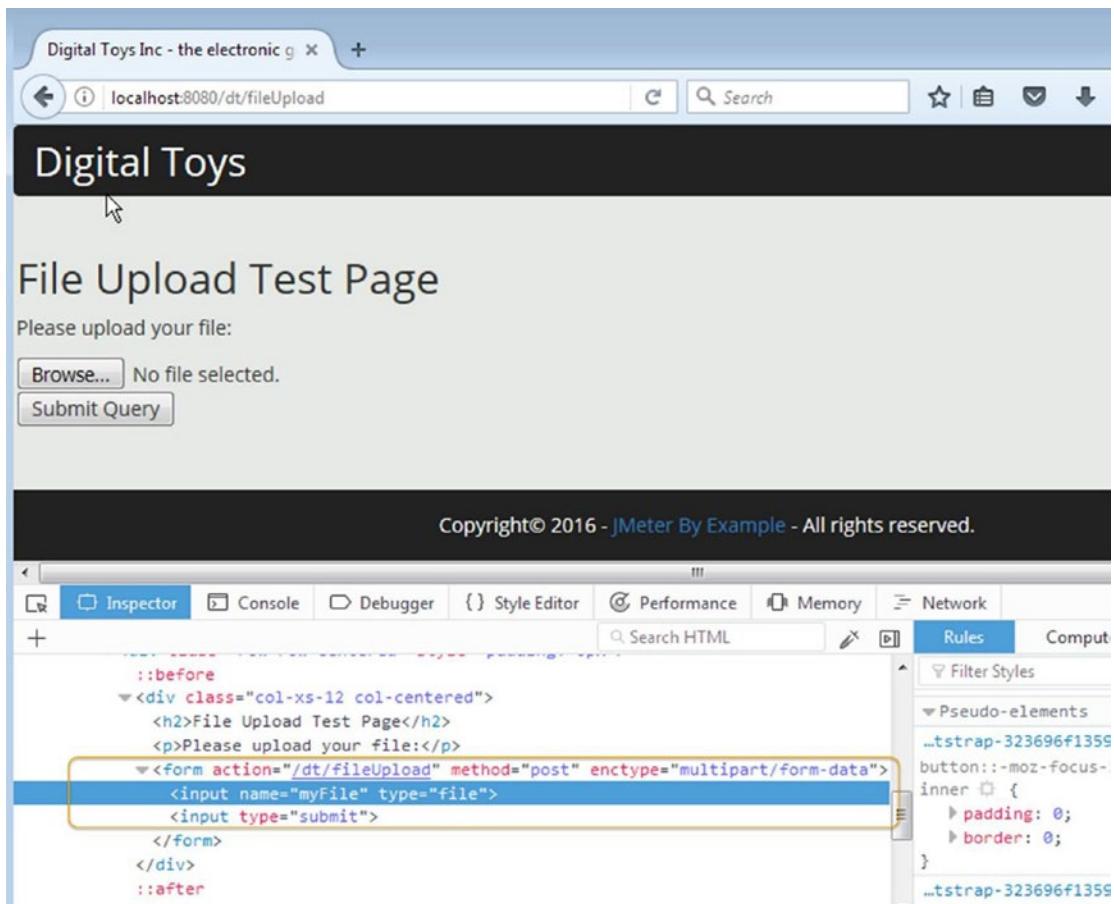


Figure 5-101. File load HTML variable

JMeter provides a few additional configuration points that can be used based on specific requirements.

- Proxy Server (if an application being tested reached through the proxy server)
- Embedded Resources for HTML Files
- Source Address
- Option Tasks

Proxy Server

JMeter provides a simple way to specify the proxy server details for the *HTTP Request* sampler. This property is only applicable to the current request. You can specify the proxy server name and the port number if needed. In addition, you may also need to specify a username and password.

Embedded Resources for HTML Files

When you use this property, the JMeter will parse the HTML file and send HTTP/HTTPS requests for all the embedded images, Java applets, JavaScript files, CSSs, etc., referenced in the file.

When you select this property, JMeter prompts you to enter these properties.

- **Use concurrent pool:** Use a pool of concurrent connections to get embedded resources.
- **Size:** Pool size for concurrent connections used to get embedded resources.

The URLs must match. This must be a regular expression that is used to match against any embedded URLs found. If you only want to download embedded resources from <http://www.yahoo.com/>, you can use the expression: http://www\.\.yahoo\.com/\.*

Let's illustrate this by the following example.

Follow these steps or download [EmbeddedResourceTestPlan.jmx](#).⁴²

1. Create a test plan and give it a meaningful name, such as Embedded Resource Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /dt, and **Method** as GET.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 5-102).

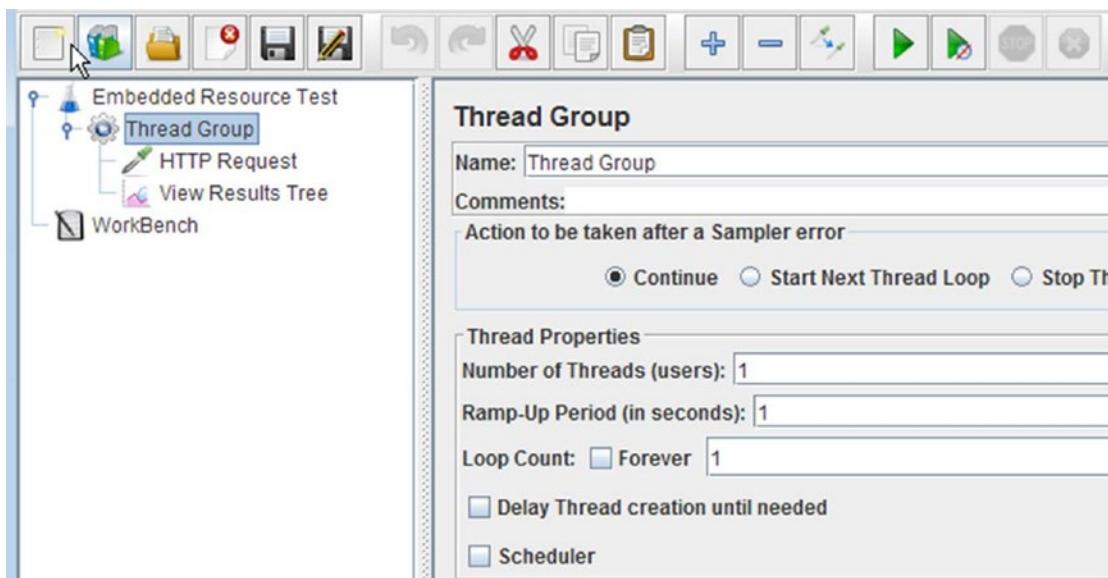


Figure 5-102. Embedded resource test

⁴²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/EmbeddedResourceTestPlan.jmx

5. Save the test plan.
6. Run the test.

The results will be similar to those shown in Figure 5-103.

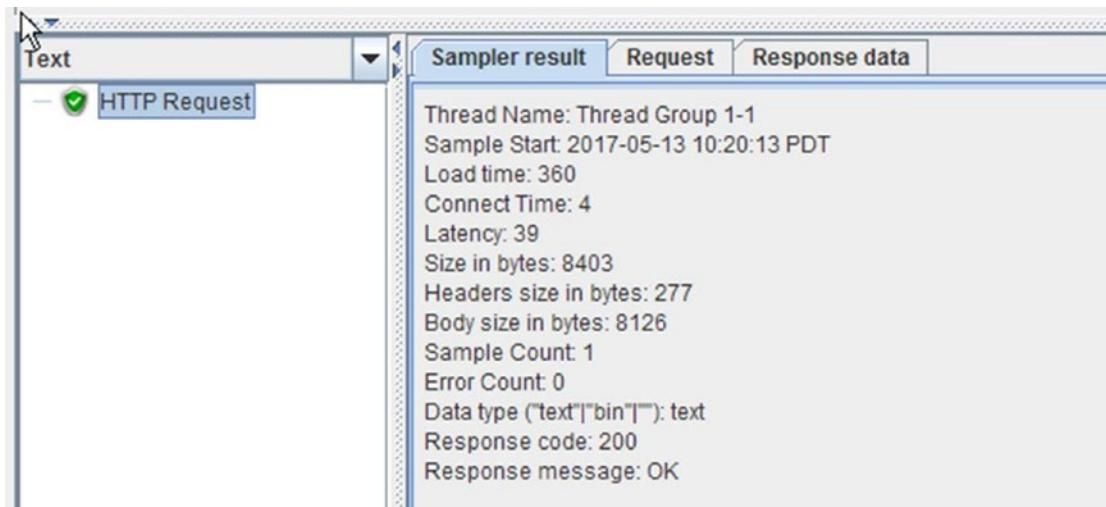


Figure 5-103. View Results Tree without an embedded resource

7. Click on HTTP Request and then click on the Advanced tab. Under Embedded Resources from HTML Files, select the checkbox for **Retrieve All Embedded Resources**
8. Save the test plan.

Run the test again and you will see a lot of requests, as shown in Figure 5-104.

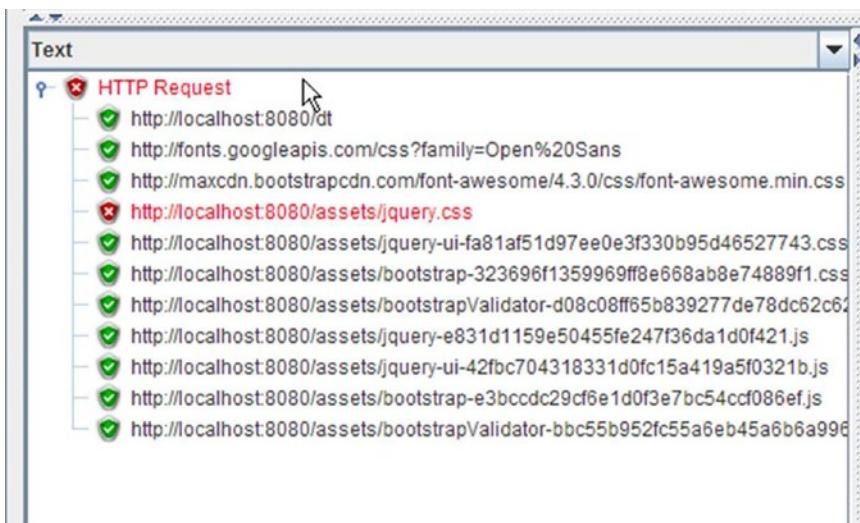


Figure 5-104. View Results Tree with embedded resource

There is one JMeter property added after the 2.1.2 release, called `HTTPResponse.parsers`. This can be separated with a space, as in `htmlParser wmlParser`. For each ID found, JMeter checks two more properties.

- **id.types:** A list of content types
- **id.className:** The parser used to extract the embedded resources

If the `HTTPResponse.parser` property is not set, then JMeter reverts to the previous behavior, i.e., only text/HTML responses will be scanned.

Source Address

This property is only used with HTTP requests that use the `HTTPClient` implementation. This property is used to enable IP spoofing.

IP spoofing is a technique used to gain unauthorized access to machines. In this, the attacker impersonates another machine by modifying the packet header with a spoofed source IP address.

If the property `httpclient.localaddress` is defined, it is used for all `HttpClient` requests.

This property is very rarely used under specific scenarios.

Option Task

This has two properties:

- **Use as Monitor:** This will be used with the Monitor Results listener.
- **Save Response as MD5 Hash?:** By using this property, the response will be encoded in MD5 hash code.

Let's look at the following example.

Follow these steps or download `MD5TestPlan.jmx`.⁴³

1. Create a test plan and give it a meaningful name, such as `MD5 Hash Code Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Sampler`. Add HTTP Request. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, **Path** as `/dt`, and **Method** as `GET`.
4. Click on the **Advanced** tab and select the **Save Response as MD5 Hash?** checkbox.
5. Click on Thread Group and go to `Edit > Add > Listener`. Add View Results Tree (see Figure 5-105).

⁴³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/samplers/MD5TestPlan.jmx

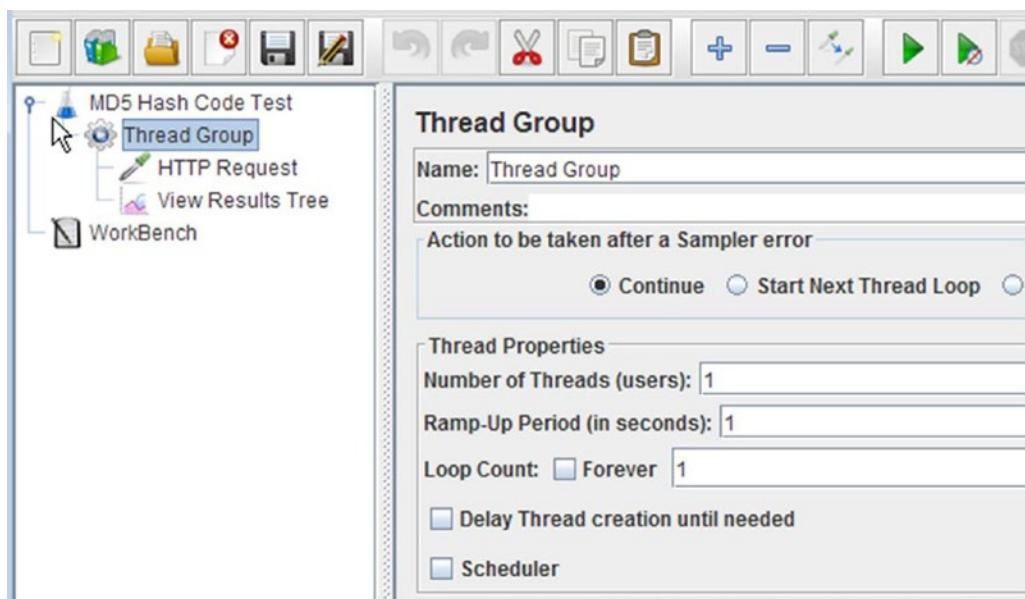


Figure 5-105. MD5 hash code test

6. Save the test plan.
7. Run the test.

The results will be similar to those shown in Figure 5-106.

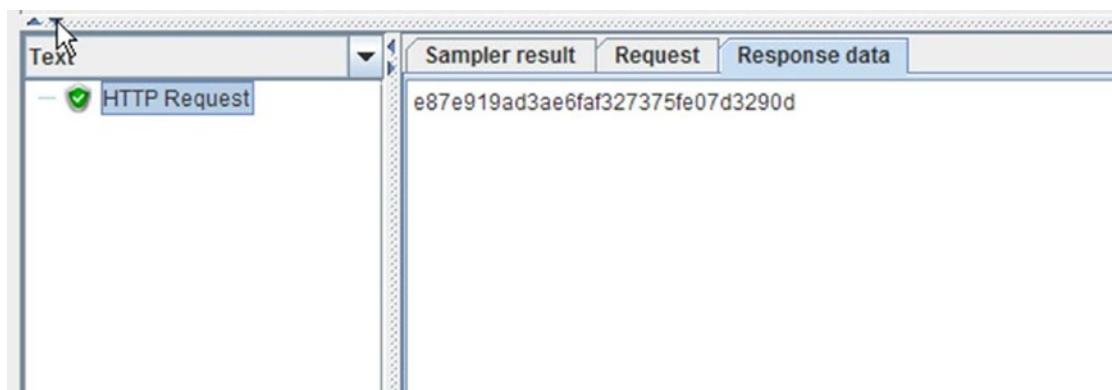


Figure 5-106. MD5 hash code

Note Use HTTP(S) Test Script Recorder to record the user actions and add HTTP Request Defaults and Config Element to capture the common parameters for all successive requests.

Assertions

The primary purpose of *assertions* is to validate the server response and decide if the test passed or failed.

The response to a sampler contains code to indicate success or error. But this is at the protocol level. JMeter assertions provides a mechanism to validate the content of the response. The specific JMeter assertions you use depend on the response format. The validation check is configurable based on the assertions type.

JMeter assertions can be a child element of a test plan, thread group, controller, or sampler. Assertions will apply to all the samplers under the scope of its parent. For example, if an assertion is a child element of a thread group, then it applies to all the samplers under the thread group. Sometimes, it makes more sense to have specific assertions for each sampler. JMeter provides the flexibility to add multiple assertions.

Assertions are used to validate the test script. Since they consume CPU and memory resources, you should remove or disable them before running the performance test to gather statistics.

The Assertion Results listener is a special listener intended specifically for viewing the results of assertions.

In the following examples, you will be using the View Results Tree listener during the test script development. However, it will be removed before you have to gather the final performance numbers.

Tip Use assertions and the Assertion Results listener with discretion, as they consume CPU and memory.

Response Assertion

The Response Assertion provides various options to validate the response from a sampler (see Figure 5-107).

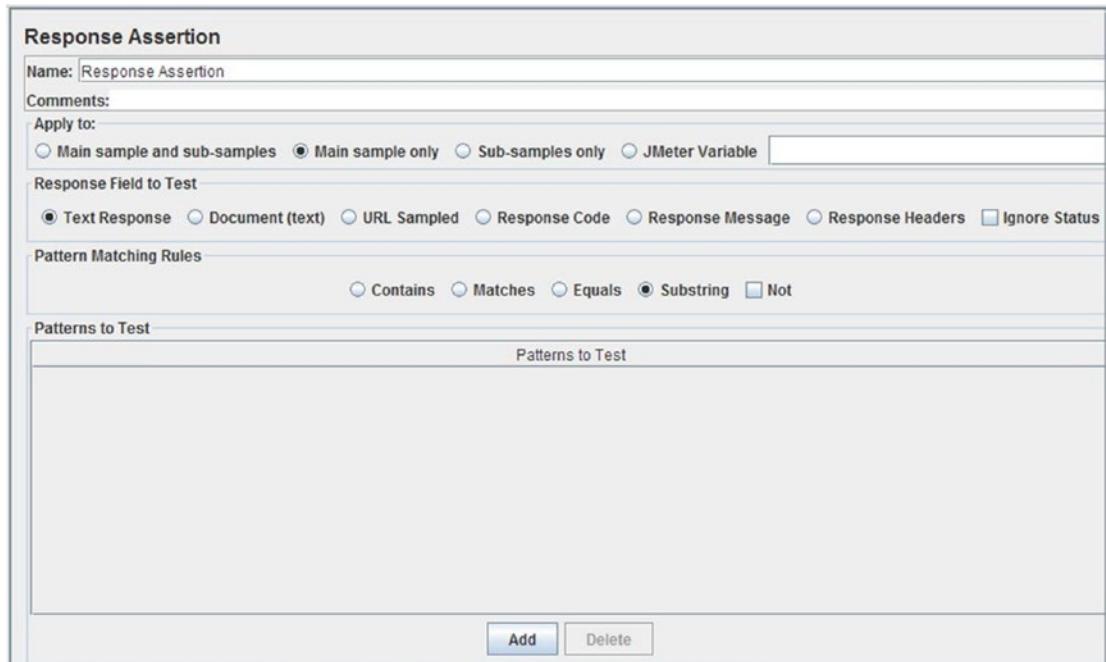


Figure 5-107. Response Assertion console

There are four categories of properties provided by JMeter assertions, discussed next.

Apply to Property

This option specifies where to apply the assertions: Main sample and sub-samples, main sample only, Sub-samples only, and JMeter variable. Sometimes a sampler may generate a redirected URL, which in turn will appear as a sub-sample and it is possible to apply assertions to these sub-samples by using Sub-Samples Only option. If you are using variables inside the test script, you can use JMeter variable option to validate it.

Response Field to Test Property

The assertion can apply to any of the following fields in the response:

- Text Response
- Document (Text)
- URL Sampled
- Response Code
- Response Message
- Response Headers
- Ignore Status

The responses with 4XX and 5XX statuses will be unsuccessful.

When Ignore Status is selected, the status code of the response is ignored and the response status is forced to be successful before evaluating the assertion.

Pattern Matching Rules Property

Table 5-3 lists the pattern matching rules.

Table 5-3. Pattern Matching Rules

Field	Uses	Description
Contains	Regular expression	True if the response contains a sub-string that matches the regular expression configured in the Pattern to Test field.
Matches	Regular expression	True if the response matches the regular expression configured in the Pattern to Test field.
Equals	Case-sensitive comparison of text	True if the response equals the text configured in the Pattern to Test field.
Sub-string	Case-sensitive comparison of text	True if the response has a sub-string in the text configured in the Pattern to Test field.

Enabling the Not checkbox negates the operator. For example, select Not and Equals to make the rule Not Equals.

Refer to the ORO covering Perl5 regular expressions⁴⁴ and also look at the JMeter User Manual for more information.⁴⁵

Pattern to Test Property

This works closely with the Pattern Matching Rules field described previously. This is a regular expression pattern or plain text, depending on the rule selected.

Take a look at the following example to understand how to utilize these configuration parameters.

Follow these steps or download SamplerAssertionsTestPlan.jmx.⁴⁶

1. Create a test plan and give it a meaningful name, such as Sampler Assertions Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Test Plan and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
4. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signIn and **Method** as POST. Add the parameters, **Name/Value** as email/user1@dt.com and password/user1.
6. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Text ResponseContains. Configure **Apply To** by selecting Main samples only, set **Response Field to Test** by selecting Text Response, set **Pattern Matching Rule** by selecting Contains, and set **Pattern To Test** to ((?:Good Shot Camera)). Note that the landing page has products with the name “Good Shot Camera” (see Figure 5-108).

⁴⁴<https://www.savarese.org/oro/docs/OROMatcher/Syntax.html#Perl5Expressions>

⁴⁵http://jmeter.apache.org/usermanual/regular_expressions.html

⁴⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/assertions/SamplerAssertionsTestPlan.jmx

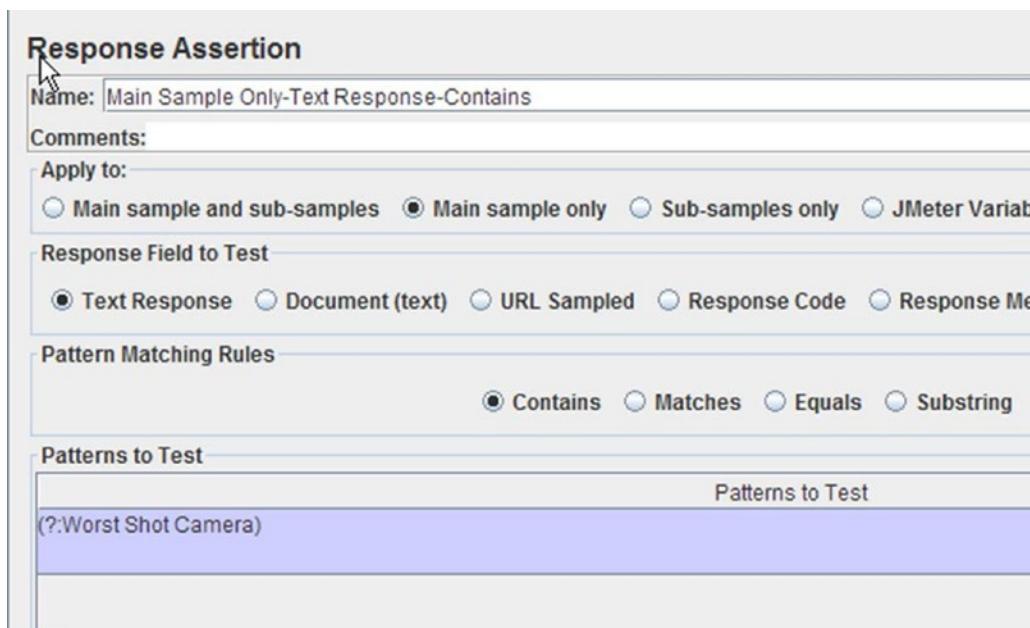


Figure 5-108. Main sample only text response contains

- Click on HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Response CodeEquals Not & Ignore Status. Configure **Apply To** by selecting Main samples only, set **Response Field to Test** by selecting Response Code, enable the Ignore Status checkbox, set the **Pattern Matching Rule** by selecting Equals, and select the Not checkbox. For **Pattern To Test**, add 500. (You have selected the Ignore Status checkbox, which states that ignore the URL status is successful; this will not be an ideal condition with the response code set to 5XX or 4XX) (see Figure 5-109).

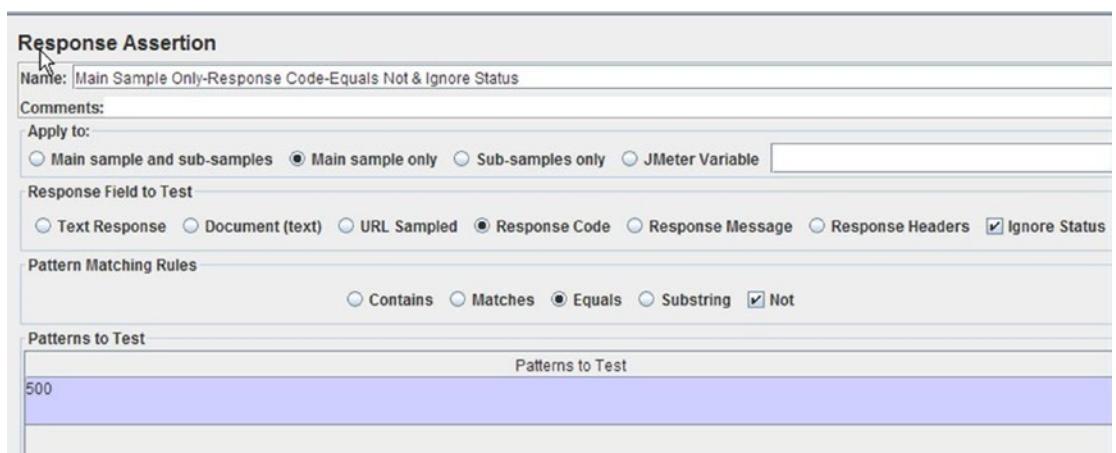


Figure 5-109. Main sample only, response code equals not ignore status

- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Text ResponseContains Not & Ignore Status. Configure **Apply To** as Main samples only and **Response Field to Test** as Text Response. Select the **Ignore Status** checkbox. Set the **Pattern Matching Rule** to Contains and then select the **Not** checkbox. Set the **Pattern To Test** to (?::URL not found) (see Figure 5-110).

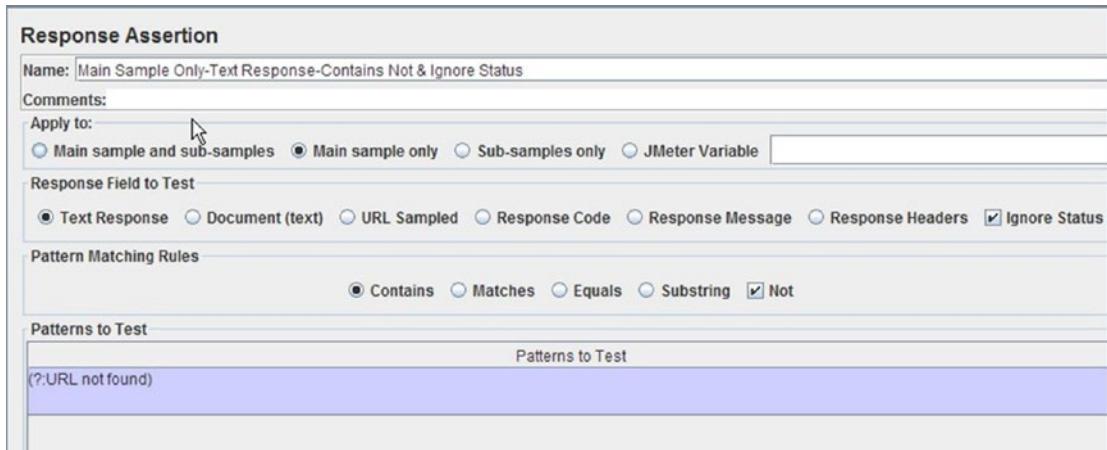


Figure 5-110. Main sample only, text response contains not ignore status

- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Document (text)-Substring. Configure **Apply To** as Main samples only, **Response Field to Test** as Document (text), **Pattern Matching Rule** as Substring, and **Pattern To Test** as <title>Digital Toys Inc - the electronic goods e-store.</title> (see Figure 5-111).

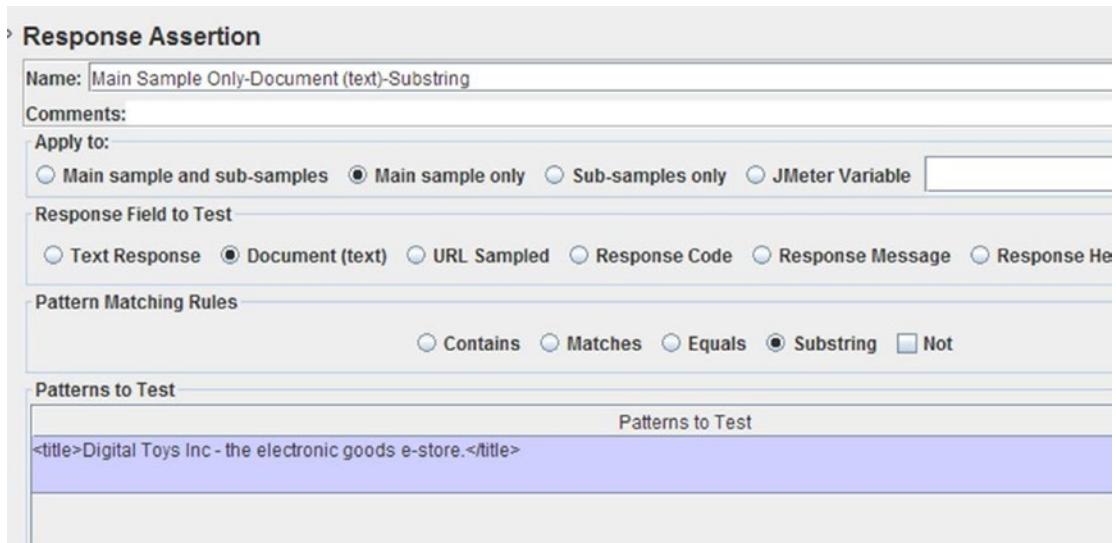


Figure 5-111. Main sample only document text sub-string

- Click on HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Give it a name such as Sub-Samples Only-URL SampledSubstring Not. Configure Apply To as Sub-samples only, Response Field to Test as URL Sampled, and Pattern Matching Rule as Substring. Select the Not checkbox and set Pattern To Test to Error (see Figure 5-112).

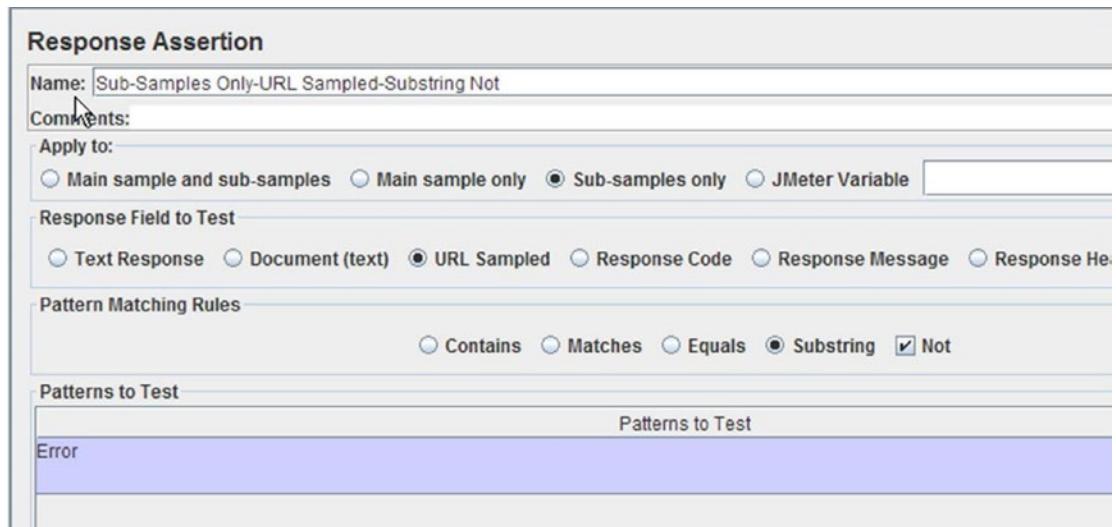


Figure 5-112. Sub-samples only URL sampled sub-string Not

11. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure the **Path** as dt/index and the **Method** as GET.
12. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name as Main Sample Only-Response CodeEquals. Configure **Apply To** as Main samples only, **Response Field to Test** as Response Code, **Pattern Matching Rule** as Equals, and **Pattern To Test** as 200 (see Figure 5-113).

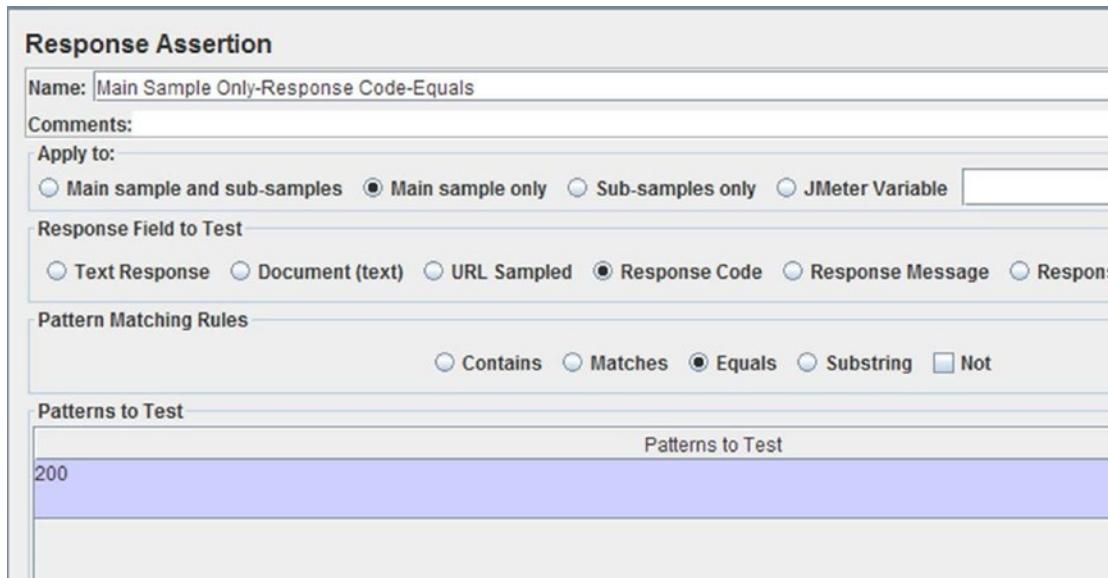


Figure 5-113. Main sample only, response code equals 200

13. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Response Message-Equals. Configure **Apply To** as Main samples only, **Response Field to Test** as Response Message, **Pattern Matching Rule** as Equals, and **Pattern To Test** as OK (see Figure 5-114).

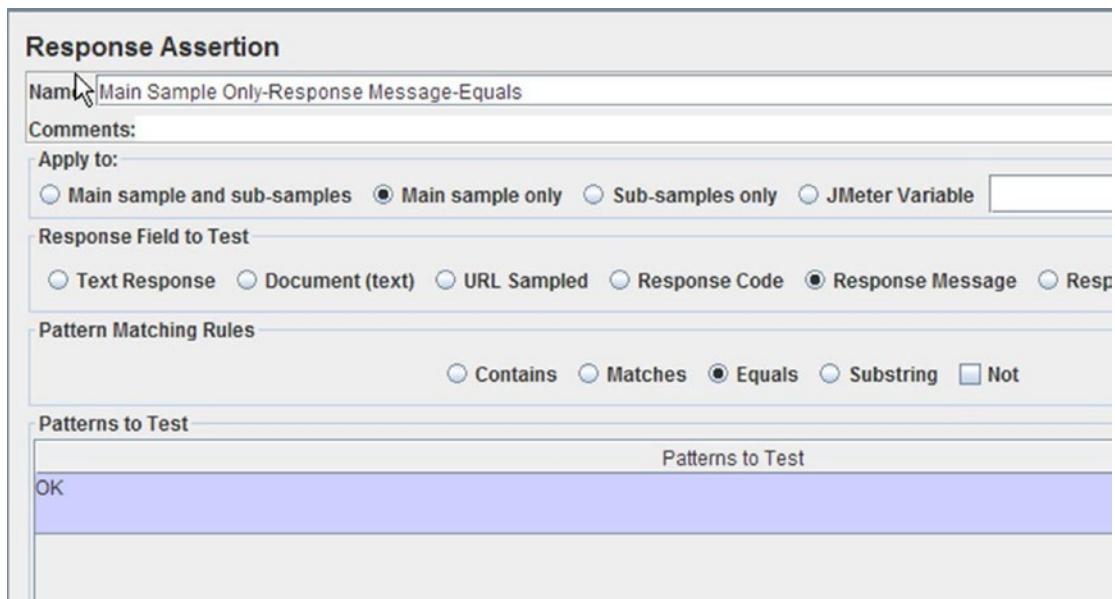


Figure 5-114. Main sample only, response message equals OK

- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample Only-Response Headers-Contains. Configure **Apply To** as Main samples only, **Response Field to Test** as Response Headers, **Pattern Matching Rule** as Contains, and **Pattern To Test** as (1?)HTTP/1.1 200 OK (see Figure 5-115).

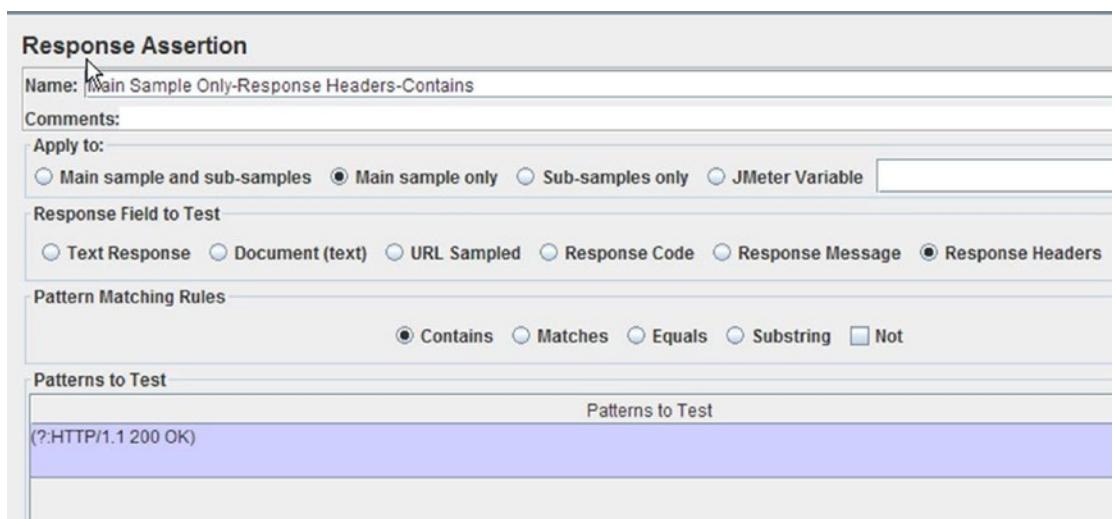


Figure 5-115. Main sample only, response headers contains

15. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signOut and **Method** as HEAD.
16. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Give it a name such as Main Sample and Sub-SamplesText Response-Matches Not. Configure **Apply To** as Main samples and sub-samples, **Response Field to Test** as Text Response, and **Pattern Matching Rule** as Matches. Select the **Not** checkbox set the **Pattern To Test** to (?:Good Shot Camera) (see Figure 5-116).

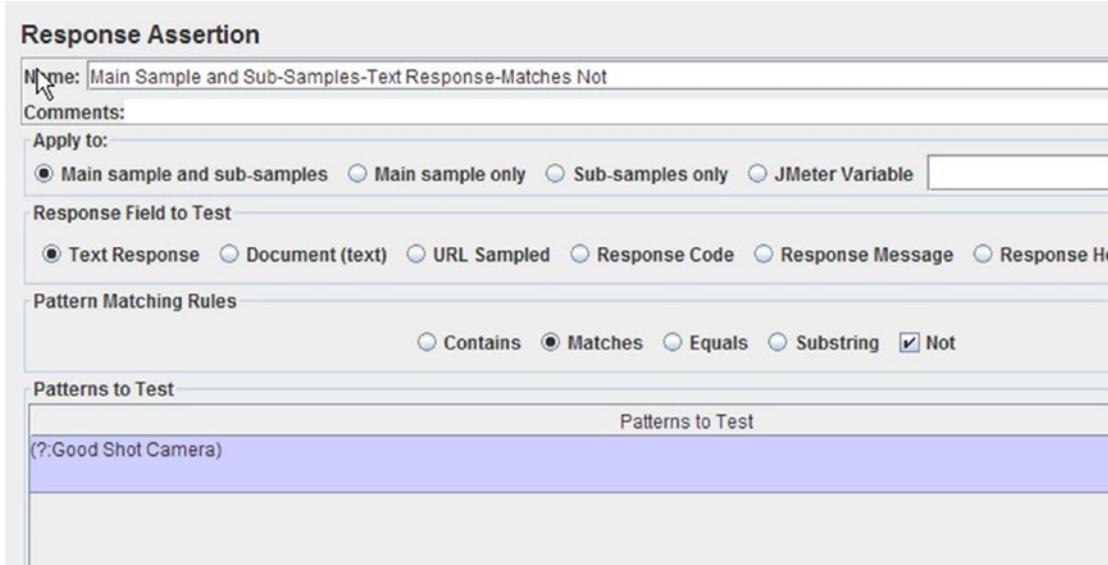


Figure 5-116. Main sample and sub-samples text response, matches not

17. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
18. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add Assertion Results (see Figure 5-117).

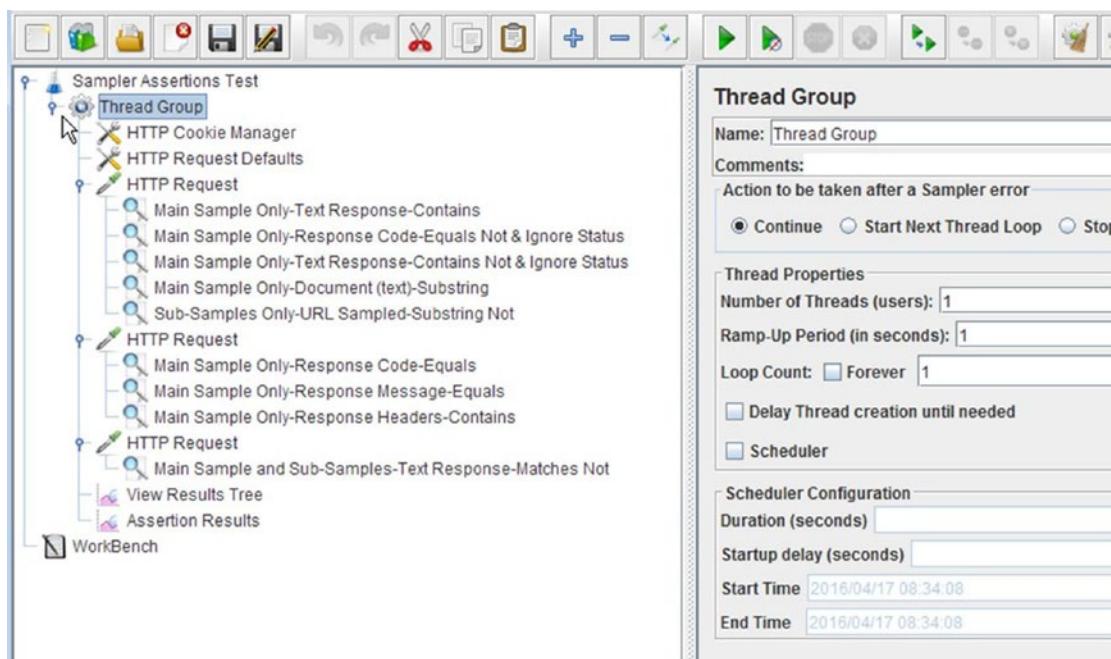


Figure 5-117. Sampler assertions test

19. Save the test plan.
20. Run the test.

The results will be similar to those shown in Figure 5-118.

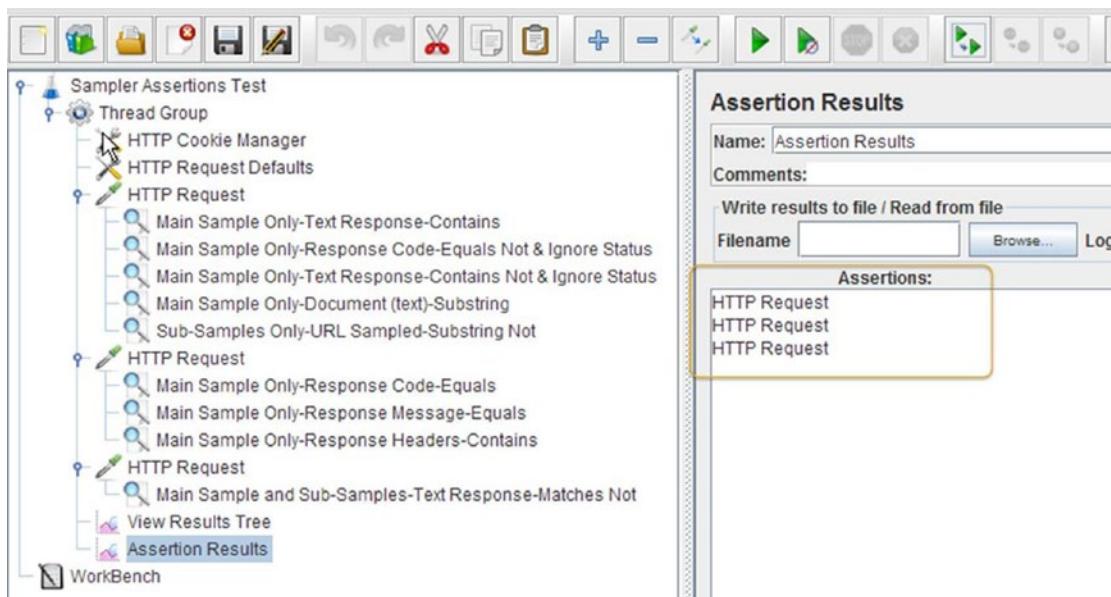


Figure 5-118. Pass test

Let's look at an example where the assertion fails.

Follow these steps.

1. Open SamplerAssertionsTestPlan.jmx.
2. Click on the first assertion called Main Sample Only-Text Response-Contains and update as (?:Worst Shot Camera).
3. Save the test plan.
4. Run the test. It should fail (see Figure 5-119).

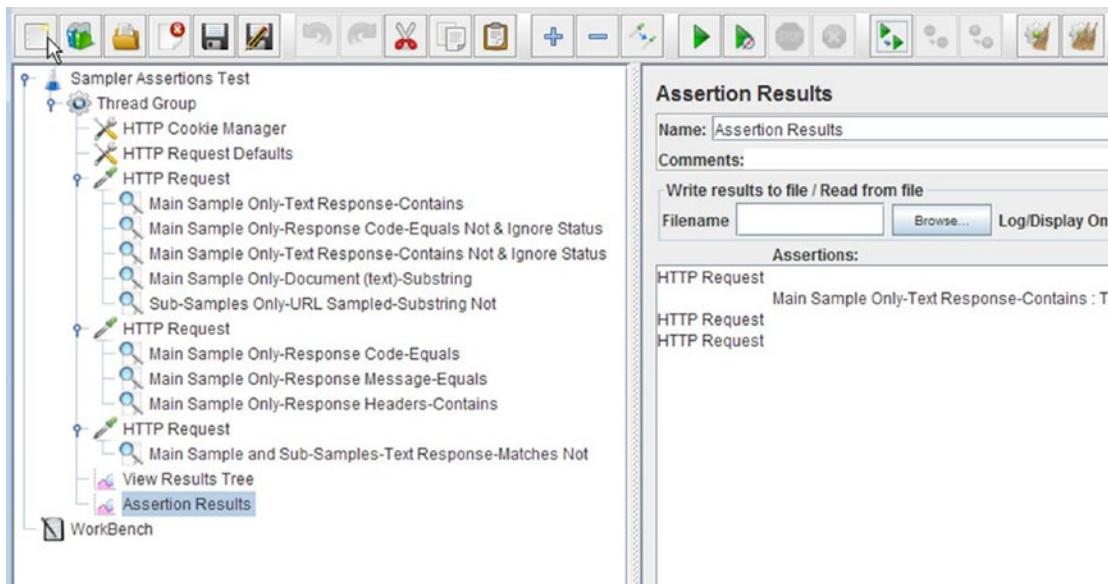


Figure 5-119. Fail test

Listener

Listeners capture and process the response from the server. Performance testing requires two kinds of listeners. During the test script development, you need to capture and display the entire server response for verifying that the output meets the functional specifications. When the performance test scripts are executed, you need the aggregate results and metrics for the duration of the test execution. You also need listeners to be able to store these results into an external file for later use.

JMeter Listeners can be configured to capture the required information by selecting individual fields provided by the JMeter Listener. The test plan can have more than one listener and they are executed after all the other types of elements in the test plan.

Listeners consume memory and CPU time to perform I/O and to process the results and generate reports. In Master/Slave mode, it is important to choose the right listeners and as few as possible, because slave nodes will write data back to the master node and overall it will slow down processing or dry out memory.

As of JMeter version 3.0, the following listeners are available, but only a few are used frequently:

- *Aggregate Graph*
- *Aggregate Report*
- *Assertion Results*
- *Backend Listener*
- *BeanShell Listener*
- *BSF Listener*
- *Comparison Assertion Analyzer*
- *Generate Summary Results*
- *Graph Results*
- *JSR233 Listener*
- *Mailer Visualizer*
- *Monitor Results*
- *Response Time Graph*
- *Save Responses to a File*
- *Simple Data Writer*
- *Summary Report*
- *View Results in Table*
- *View Results in Tree*

Most of these listeners—except for *Backend Listener*, *BeanShell Listener*, *BSF Listener*, *Generate Summary Results*, *JSR233 Listener*, and *Save Responses to a File*—can be configured using the *Configure* button on the right side.

Listeners allow you to save data in CSV/XML formats. Bear in mind that XML files consume more memory than CSV files (see Figure 5-120).

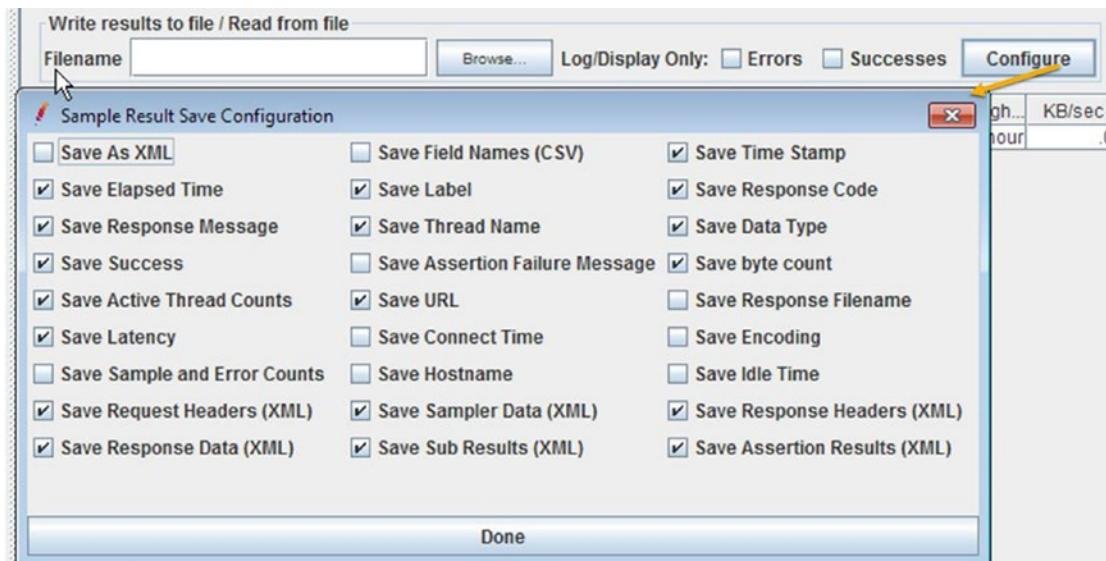


Figure 5-120. Listener configuration

Note All JMeter Listeners store similar data to the external file, but GUI presentation is different.

All of the configuration parameters can be configured through the `jmeter.properties` file.
Set the following property to either CSV or XML:

```
# legitimate values: xml, csv, db. Only xml and csv are currently supported. #jmeter.save.saveservice.output_format=csv
```

Other configuration fields can be set in `jmeter.properties`, and all these parameters can be found with the prefix `jmeter.save.saveservice`.

```
# Timestamp format - this only affects CSV output files  
# legitimate values: none, ms, or a format suitable for  
SimpleDateFormat  
#jmeter.save.saveservice.timestamp_format=ms
```

JMeter supports additional sample variables you can specify, as shown here. They will appear in the CSV/XML file as additional fields.

```
# Optional list of JMeter variable names whose values are to be saved in the result data  
files. # Use commas to separate the names. For example:  
#sample_variables=SESSION_ID,REFERENCE
```

JMeter Listeners follow the SimpleDateFormat.⁴⁷ JMeter first tries to parse this as a long integer. Failing that, it will try the following formats.

yyyy/MM/dd HH:mm:ss.SSS yyyy/MM/dd HH:mm:ss yyyy-MM-dd HH:mm:ss.SSS yyyy-MM-dd HH:mm:ss
MM/dd/yy HH:mm:ss (this is for compatibility with previous versions; it is not recommended as a format)

Matching is strict (non-lenient). JMeter 2.8 and earlier used lenient mode, which could result in timestamps with incorrect dates (times were usually correct).

The most commonly used listeners are *View Results Tree*, *View Results in Table*, and *Aggregate Report*. Let's discuss these in the following sections.

View Results Tree

GUI Mode

The *View Results Tree* listener shows responses in a tree-like structure, thereby allowing users to see response data (content), sample results, and requests. It also shows the time taken by the request to get the response.

The HTTP protocol implementation (such as HttpClient3.1, HttpClient4, Java, or blank) adds an HTTP header with the name User-Agent. The request panel of the View Results Tree does not show the headers that may have been added by the HTTP protocol implementation.

JMeter provides three panels with most of the listeners, namely sample results, requests, response data, and various filters to check the response data.

- The Sample Results panel has two views: Raw and Parsed
- The Request panel also has two views: Raw and HTTP
- The Response Data panel has search enabled and you can find data using regular expressions as well as perform case-sensitive searches

Let's illustrate this with the following example.

Follow these steps or download `ViewResultsTreeTestPlan.jmx`.⁴⁸

1. Create a test plan and give it a meaningful name, such as **View Results Tree Listener Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add **HTTP Header Manager**.
4. Click on Thread Group and go to **Edit > Add > Sampler**. Add **HTTP Request**. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
5. Click on **HTTP Request** and go to **Edit > Add > Assertions**. Add **Response Assertion**. Configure **Response Field to Test** as **Response Code**, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on Thread Group and go to **Edit > Add > Listener**. Add **View Results Tree** (see Figure 5-121).

⁴⁷<http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html>

⁴⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/listeners/ViewResultsTreeTestPlan.jmx

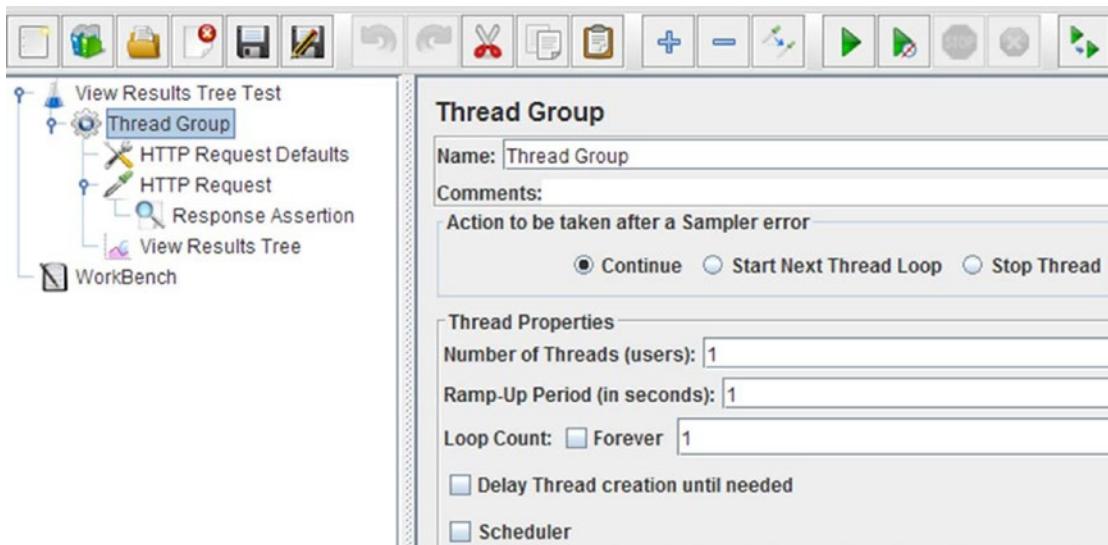


Figure 5-121. View results tree listener test

7. Save the test plan.
8. Run the test.

The results will be similar to those shown in Figure 5-122.

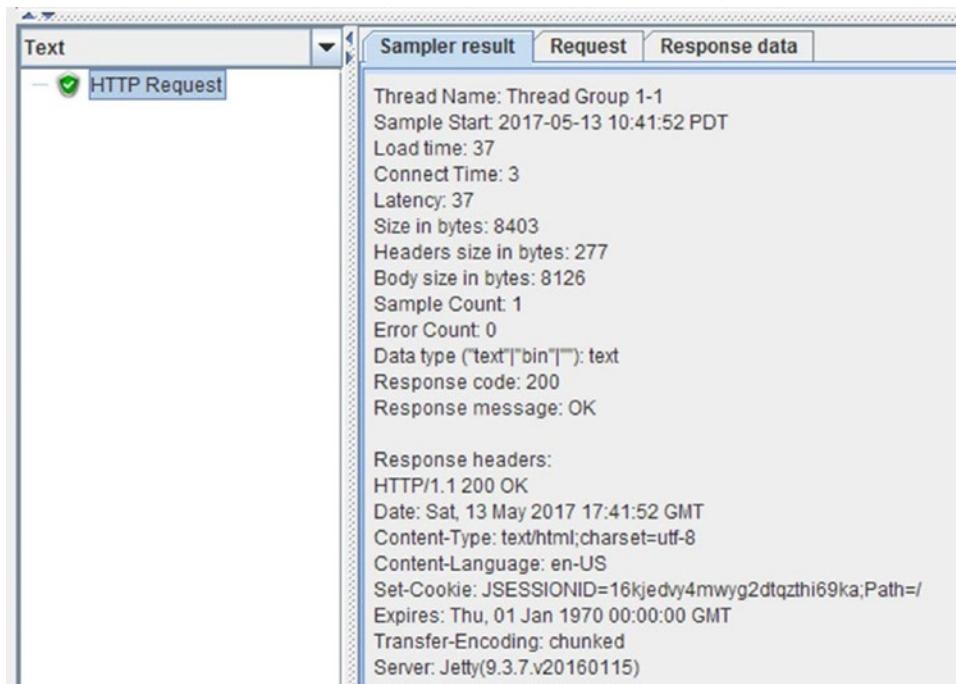


Figure 5-122. View Results Tree sampler results, raw

9. The View Results Tree has an option to search for the request. You enter the HTTP request name in the search box and click on Search button. The matched request will appear in red (see Figure 5-123).

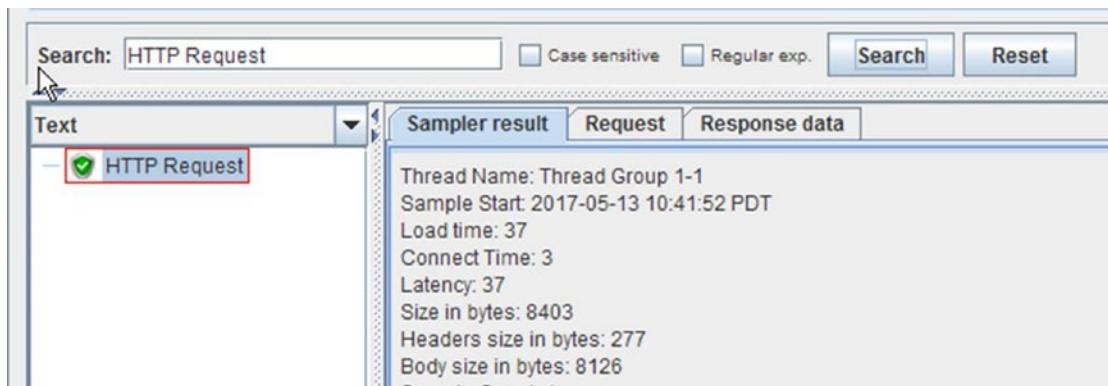


Figure 5-123. Search request in View Results Tree

10. Look at the sampler results, parsed format. It has the same information as in raw format but in a tabular format and is more human-readable (see Figure 5-124).

Text		Sampler result	Request	Response data
–	HTTP Request			
		Thread Name	Thread Group 1-1	
		Sample Start	2017-05-13 10:41:52 PDT	
		Load time	37	
		Latency	37	
		Size in bytes	8403	
		Headers size in bytes	277	
		Body size in bytes	8126	
		Sample Count	1	
		Response header	Value	
		HTTP/1.1 200 OK		
		Date	Sat, 13 May 2017 17:41:52 GMT	
		Content-Type	text/html; charset=utf-8	
		Content-Language	en-US	
		Set-Cookie	JSESSIONID=16kjedvy4mwyg2dtqzthi69ka; Path=/	
		Expires	Thu, 01 Jan 1970 00:00:00 GMT	
		Transfer-Encoding	chunked	
		Server	Jetty(9.3.7.v20160115)	
		Additional field	Value	
		Type Result	HTTPSSampleResult	
		ContentType	text/html; charset=utf-8	
		Raw	Parsed	
		Scroll automatically?		

Figure 5-124. View Results Tree sampler results parsed

11. Look at the Request - Raw format as well (see Figure 5-125).

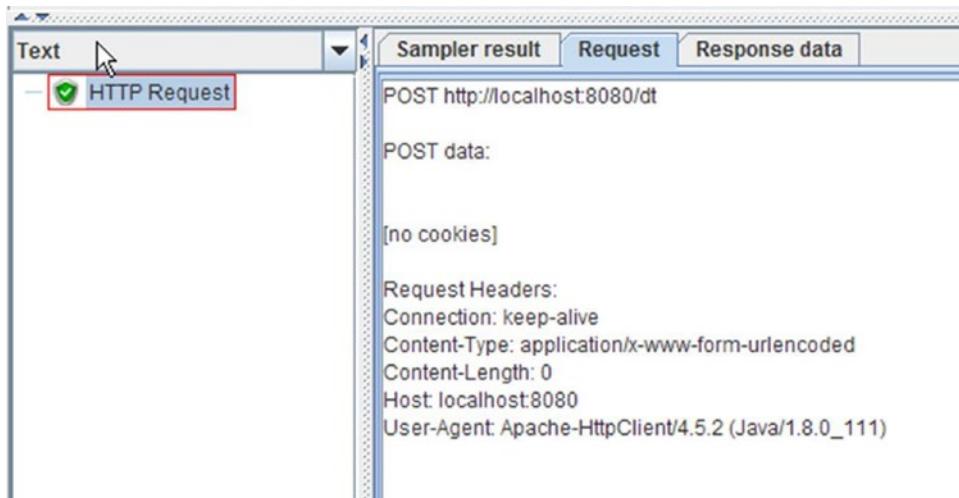


Figure 5-125. View Results Tree request. raw

12. Look at the Request - HTTP format as well. It has the same information as in raw format, but in a tabular format and is more human-readable (see Figure 5-126).

Parameter name	Value
Method	POST
Protocol	http
Host	localhost
Port	8080
Request header	Value
Connection	keep-alive
Content-Type	application/x-www-form-urlencoded
Content-Length	0
Host	localhost:8080
User-Agent	Apache-HttpClient/4.5.2 (Java/1.8.0_111)

Figure 5-126. View Results tree request HTTP

13. Look at the Response Filters field. It has search options based on response filters and works on the visible text. For each filter type, there will be different text visible on the panel, so make sure that if you are searching for HTML-related text to open the response data with an HTML filter. The regular expression uses the Java engine (not an ORO engine like the Regular Expression Extractor or RegExp Tester view) (see Figure 5-127).

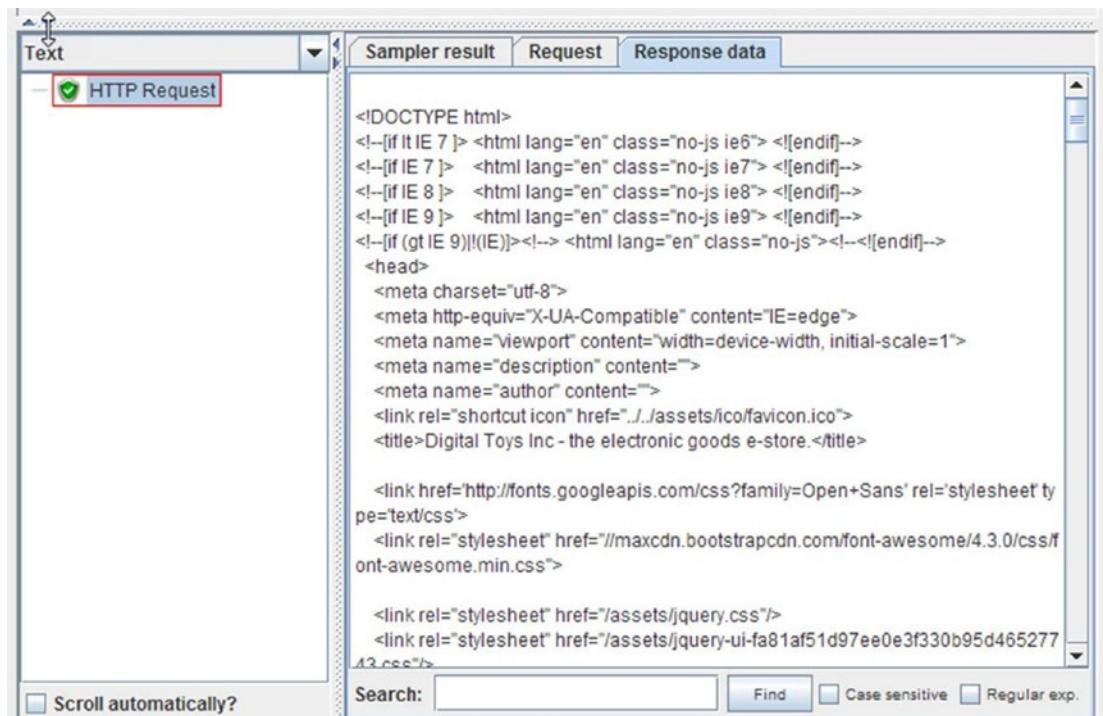


Figure 5-127. View Results Tree response data filters

If you want to change the default size of the response data, use the following `jmeter.properties` property.

```
# Maximum size of Document that can be parsed by Tika engine; default=10 * 1024 * 1024
# (10MB) # Set to 0 to disable the size check
#document.max_size=0
```

Note By default, only response data up to 200K is visible in the panel.

This listener also provides an option to save results to a file (see Figure 5-128).

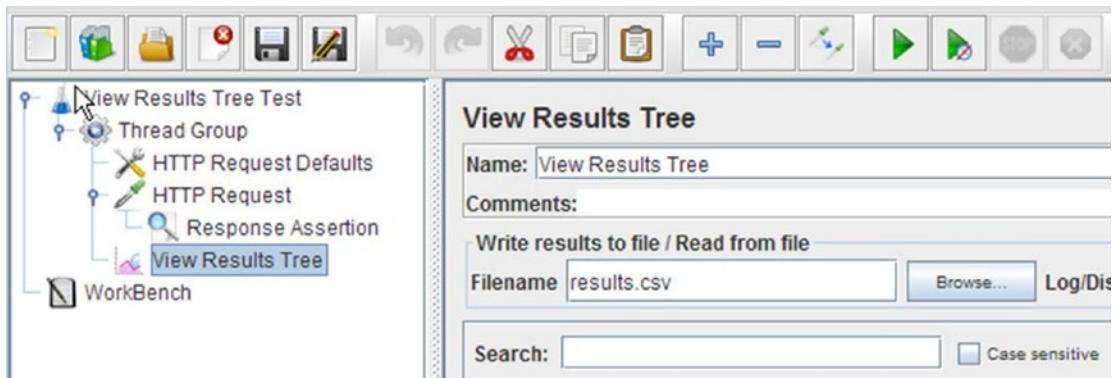


Figure 5-128. *Filename*

Provide the name of the file and it will save the file inside the `/bin` directory of JMeter. If you provide a full path, the file will be saved inside that directory. The generated file is a comma-separated file with the default configuration.

Look at the contents of the file. Contents may differ based on your configuration of listeners, but the original objective is the save the results to a file.

```
c:\apache-jmeter-3.0\bin>type results.csv
1454179982085,3355,HTTP Request,200,OK,Thread Group 1-2,text,true,279689,10,10,456
1454179982187,3418,HTTP Request,200,OK,Thread Group 1-3,text,true,279725,9,9,472
1454179982386,3267,HTTP Request,200,OK,Thread Group 1-5,text,true,279709,8,8,468
1454179982289,3433,HTTP Request,200,OK,Thread Group 1-4,text,true,279695,7,7,476
1454179981982,3772,HTTP Request,200,OK,Thread Group 1-1,text,true,279750,6,6,467
1454179982484,3318,HTTP Request,200,OK,Thread Group 1-6,text,true,279650,5,5,465
1454179982586,3254,HTTP Request,200,OK,Thread Group 1-7,text,true,279760,4,4,470
1454179982687,3244,HTTP Request,200,OK,Thread Group 1-8,text,true,279709,3,3,461
1454179982788,3299,HTTP Request,200,OK,Thread Group 1-9,text,true,279750,2,2,468
1454179982888,3248,HTTP Request,200,OK,Thread Group 1-10,text,true,279707,1,1,458
```

Non-GUI Mode

When you are running tests in non-GUI mode and want to see results in this listener, pass the `-l` flag with the filename in the command.

```
C:\>jmeter -n -t ViewResultsTreeTestPlan.jmx -l results.jtl -j results.csv
```

```
Writing log file to: results.csv
Creating summariser <summary>
Created the tree successfully using ViewResultsTreeTestPlan.jmx
Starting the test @ Mon May 15 12:26:35 PDT 2017 (1494876395416)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 1 in 00:00:01 = 1.7/s Avg: 284 Min: 284 Max: 284 Err: 0 (0.00%)
Tidying up ... @ Mon May 15 12:26:36 PDT 2017 (1494876396167)
... end of run
```

Note By default, if you run tests in non-GUI mode, the test results are appended to an .jtl file. Make sure that you delete .jtl files appropriately.

Open the View Results Tree and from the section called Write Results to File/Read from File. Click on the Browser button and locate the file (generated using the command) and the tree view will be visible on GUI (see Figure 5-129).



Figure 5-129. View Results Tree read from file

You can filter logs based on errors or successes, or both, by using the checkbox.

If you have not set the custom fields for the results file configuration section, you will not see any output under the Request or Response data panels.

Stop JMeter and set the following parameters in the \$JMETER_HOME\bin\user.properties file.

```
# Set below parameters for saving response data for listeners
jmeter.save.saveservice.output_format=xml jmeter.save.saveservice.response_data=true
jmeter.save.saveservice.samplerData=true jmeter.save.saveservice.requestHeaders=true
jmeter.save.saveservice.url=true jmeter.save.saveservice.responseHeaders=true
```

Re-run test in non-GUI mode and open the .jtl file generated by the listener. It will show all the details just like when you have executed tests in GUI mode.

Caution View Results Tree takes lot of memory to hold results and should not be used while doing actual performance testing.

View Results In Table

GUI Mode

The *View Results in Table* listener shows responses in a table-like structure. It also shows the time taken by the request/thread to get the response. Unlike *View Results Tree*, it does not have a panel and does not show any headers or response contents.

It has the following columns in the view.

- **Start Time:** When this thread started.
- **Thread Name:** Thread group name with the thread number in the format X-X.
- **Label:** Name of HTTP request (sample).
- **Sample Time (ms):** Difference between time when request was sent and time when response has been fully received.

- **Status:** It has two statuses: 1. Success, 2. Warning. If the thread sample is not valid, it shows Warning; otherwise, it shows Success.
- **Bytes:** Total number of bytes received as a part of the response.
- **Latency:** JMeter measures the latency from just before sending the request to just after the first response is received.
- **Connect Time(ms):** JMeter measures the time it takes to establish the connection, including the SSL handshake.

Let's illustrate this with the following example.

Follow these steps or download [ViewResultsInTableTestPlan.jmx](#).⁴⁹

1. Create a test plan and give it a meaningful name, such as **View Results In Table Listener Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Config Element**. Add HTTP Header Manager.
4. Click on Thread Group and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
5. Click on HTTP Request and go to **Edit > Add > Assertions**. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on Thread Group and go to **Edit > Add > Listener**. Add View Results in Table (see Figure 5-130).

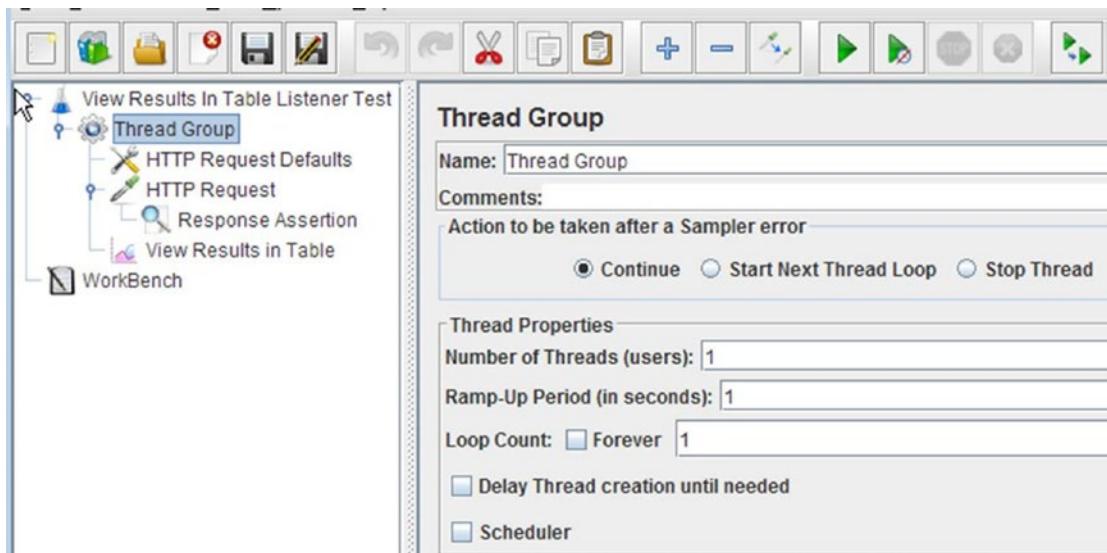


Figure 5-130. View Results in Table listener test

⁴⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/listeners/ViewResultsInTableTestPlan.jmx

7. Save the test plan.
8. Run the test.

The results will be similar to those shown in Figure 5-131.

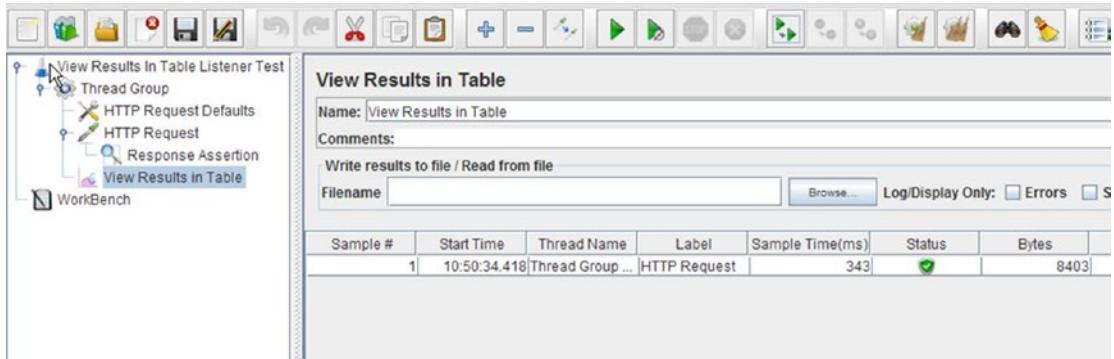


Figure 5-131. View Results in Table

Provide the name of the file and it will save the file inside the /bin directory of JMeter. If you provide full path, the file will be saved inside that directory. The generated file is a comma-separated file with the default configuration.

Non-GUI Mode

When running tests in non-GUI mode, you want to see results in this listener pass -l flag with the filename in the command.

```
C:\>jmeter -n -t ViewResultsInTableTestPlan.jmx -l results.jtl -j results.csv
```

```
Writing log file to: results.csv
Creating summariser <summary>
Created the tree successfully using ViewResultsInTableTestPlan.jmx
Starting the test @ Mon May 15 12:30:04 PDT 2017 (1494876604839)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary + 1 in 00:00:01 = 1.7/s Avg: 293 Min: 293 Max: 293 Err: 0 (0.00%)
Active: 1 Started: 1 Finished: 0
summary = 1 in 00:00:01 = 1.6/s Avg: 293 Min: 293 Max: 293 Err: 0 (0.00%)
Tidying up ... @ Mon May 15 12:30:05 PDT 2017 (1494876605614)
... end of run
```

Open View Results in Table and locate the section Write Results to File/Read from File. Click on the Browser button and locate the file (generated using the command). The tree view will be visible on GUI. It will show the connection time as 0.

You can filter logs based on errors or successes, or both, by using that checkbox.

Caution View Results in Table takes lot of memory to hold results and should not be used while doing actual performance testing.

Aggregate Report

GUI Mode

The *Aggregate Report* listener also shows the responses in a table-like structure for each differently named sampler. Similar to View Results in Table, it does not have a panel and does not show any headers or response contents. Generated results can be saved as .csv files with the use of the Save Table Data button at the bottom of this listener.

If you have multiple thread groups in your test script, it is useful to display the thread group name. This can be achieved by enabling the **Include Group Name in Label?** checkbox (by default, it's unchecked). By enabling the Save Table Header checkbox, you can save headers of the table with the generated results.

For each sampler, it shows the following columns in the view:

- **Label:** Name given to the sampler. If the **Include Group Name in Label?** checkbox is checked, then the thread group name is prefixed with the sampler name.
- **Samples:** Number of samples for a sampler. If there is more than one sampler with the same name, it will be combined to a single row and the combined sum is shown.
- **Average:** This is the average time in milliseconds for a set of results.
- **Median:** This is the median time in milliseconds.
- **90% Line:** This is the time in milliseconds below which 90% of the samples fall. The other samples took at least as long as this.
- **95% Line:** This is the time in milliseconds below which 95% of the samples fall. The other samples took at least as long as this.
- **99% Line:** This is the time in milliseconds below which 99% of the samples fall. The other samples took at least as long as this.
- **Min:** This the shortest time in milliseconds for the sampler with the same name.
- **Max:** This the maximum time in milliseconds for the sampler with the same name.
- **Error %:** Percent of requests with errors; it may be in fractions.
- **Throughput:** Calculated as requests/unit of time. It is the time difference between the end of the last sample and the start of the first sample, including any gaps between samples. It is expressed as number of requests/total time.
- **KB/sec:** The response received in kilobytes per second.

Let's illustrate this by the following example.

Follow these steps or download `AggregateReportTestPlan.jmx`.⁵⁰

1. Create a test plan and give it a meaningful name, such as `Aggregate Report Listener Test`.
2. Click on Test Plan and go to `Edit > Add > Threads (Users)`. Add Thread Group.
3. Click on Thread Group and go to `Edit > Add > Config Element`. Add HTTP Header Manager.

⁵⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/listeners/AggregateReportTestPlan.jmx

4. Click on Thread Group and go to Edit > Add > Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
5. Click on HTTP Request and go to Edit > Add > Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on Thread Group and go to Edit > Add > Listener. Add Aggregate Report (see Figure 5-132).

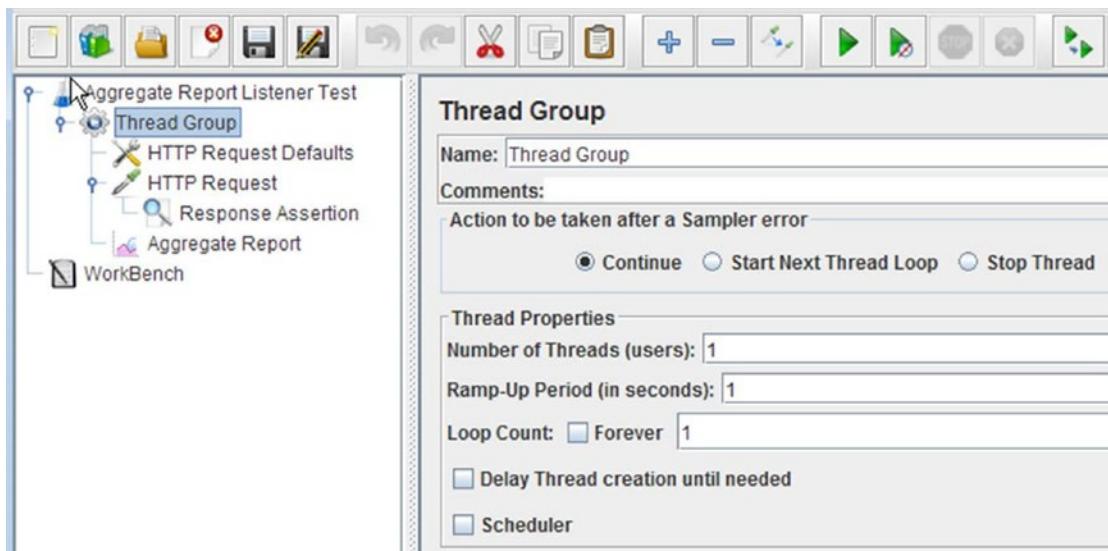


Figure 5-132. Aggregate report listener test

7. Save the test plan.
8. Run the test.

The results will be similar to those shown in Figure 5-133.

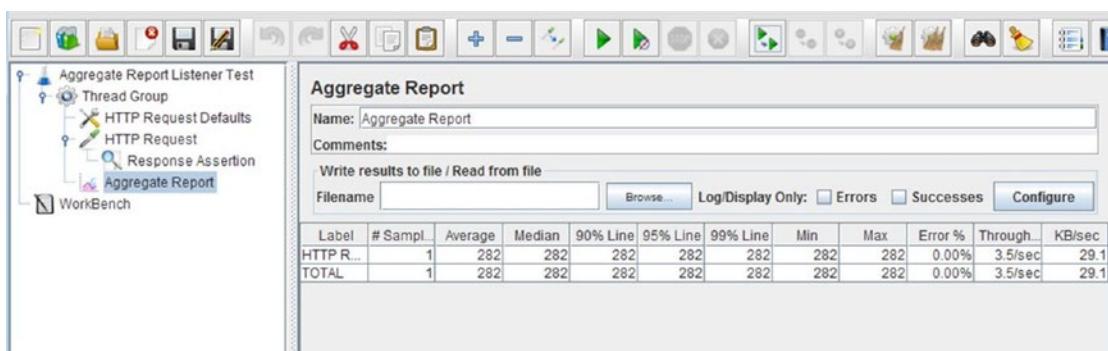


Figure 5-133. Aggregate report

Provide the name of the file and it will save the file inside the /bin directory of JMeter. If you provide full path, the file will be saved inside that directory. The generated file is a comma-separated file with the default configuration.

After opening the generated XML file, you can see an XML representation with a default configuration. Searching for `httpSample` shows the HTTP request used in the test.

Non-GUI Mode

When running tests in non-GUI mode, if you want to see results in this listener, pass the `-l` flag with the filename in the command.

```
C:\> jmeter -n -t AggregateReportTestPlan.jmx -l result.jtl -j result.log
Writing log file to: result.log
Creating summariser <summary>
Created the tree successfully using AggregateReportTestPlan.jmx
Starting the test @ Mon May 15 12:34:26 PDT 2017 (1494876866750)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 1 in 00:00:01 = 1.7/s Avg: 289 Min: 289 Max: 289 Err: 0 (0.00%)
Tidying up ... @ Mon May 15 12:34:27 PDT 2017 (1494876867527)
... end of run
```

Open Aggregate Report and find the section Write Results to File/Read from File. Click on the Browser button and locate the file (generated using the command). The tree view will be visible on GUI.

You can filter logs based on errors or successes, or both, by using that checkbox.

Note Aggregate Report Listener no longer keeps copies of every single sample. Instead, it aggregates and keeps only the summary report. This listener is suitable for actual performance testing.

Post-Processors

After the sampler gets executed, you need a post-processor to extract useful information from the response and store the value in a variable to make it available to the next sampler. Post-processors are executed before the listener and after the sampler, as per the JMeter test plan execution order.

Regular Expression Extractor

The Regular Expression Extractor is used to extract useful information (mostly values of variables to be used in successive requests) from the response of a sampler. If it is added as a child to a thread group, then the Regular Expression Extractor is applied to all the child samplers (see Figure 5-134).

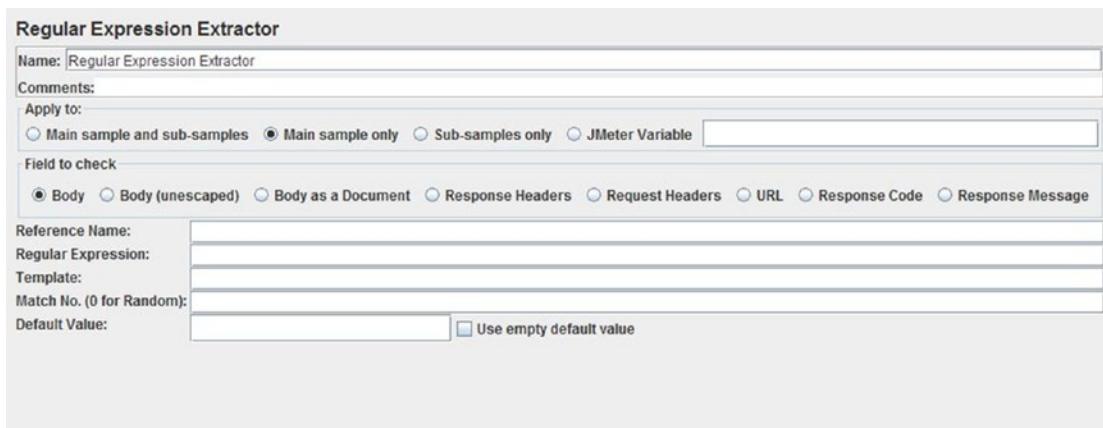


Figure 5-134. Regular expression extractor

This has the following options:

- **Apply To:** This has four options:
 - *Main samples and sub-samples*: Applies to main samples and sub-samples
 - *Main sample only*: Applies to main samples
 - *Sub-samples only*: Applies to sub-samples
 - *JMeter Variable*: Applies to the contents of the JMeter variable
- **Field to Check:** This has eight options:
 - *Body*: Body of the response (excluding headers)
 - *Body (Unescaped)*: Body of the response having HTML escape codes replaced (use caution, as it consumes additional CPU time and memory)
 - *Body as a Document*: Extract text from various types of documents
 - *Response Headers*: These are response headers that are returned in the sampler response.
 - *Request Headers*: Request headers used in the sampler (not required for non-HTTP sampler)
 - *URL, Response Code*: These are typical response codes, such as 2XX, 3XX, 4XX, and 5XX
 - *Response Message*: These are messages from the response, such as OK
- **Reference Name:** This is the variable name that will be used to store the parsed value.
- **Regular Expression:** This is the regular expression which will parse the sampler response.⁵¹

⁵¹http://jmeter.apache.org/usermanual/regular_expressions.html

- **Template:** This is used to create a string from the matches found by the regular expression. `1` refers to the first group, `2` refers to the second group, and `0` refers to the whole expression matched string.
- **Match No. (0 for Random):** This signifies which match to use in case there are multiple matches found by the regular expression.
- **Default Value:** This is used to set the default value for Reference Name in case the regular expression does not result any matches.

Let's say that you configure the Reference Name as `cardId`. JMeter will store these as follows:

- `cardId`: Original variable name or reference name
- `cardId_gn`: Where gn is the group number
- `cardId_g`: Where g is the total number of groups.

If you set the Match No. (0 for Random) to -ve then all possible matches in the sampler response data are processed.

- `cardId_matchNr`: Number of matches found. If none are found then it will be zero
- `cardId_n`: Number of strings generated by the template
- `cardId_n_gm`: gm groups for n matches
- `cardId`: Default value will be used
- `cardId_gn`: This will not be set in this case

Let's illustrate these with the following example. (For the test script example, we will be using Apply To as Main sample only and **Field to Check** as Body.)

Before starting the test, let's set up the browser for recording steps on JMeter. Set the browser and JMeter HTTP(S) test script recorder, and the global settings port to the same port.

Next, let's set up proxy in Firefox browser.

1. Open Firefox browser and select Preferences > Advanced > Network > Connection Settings > Manual Proxy Configuration. Configure **HTTP Proxy** as **localhost** and **Port** as **7070**.
2. Select **Use this Proxy Server for All Protocols**.
3. Remove anything under **No Proxy** for **text Area** and then click OK.

The Firefox browser is now set up to send HTTP requests to the JMeter proxy.

Follow these steps or download `RegularExpressionExtractorTestPlan.jmx`.⁵²

1. Create a test plan and give it a meaningful name, such as **Regular Expression Extractor Test**.
2. Click on WorkBench and go to Edit > Add > Non-Test Elements. Add **HTTP(S) Script Test Recorder**. Configure **Global Settings**, **Port** as **7070**. Select **Target Controller** as **WorkBench > HTTP(S) Test Script Recorder**.
3. Add Exclude Regular Expression As: `.*\.(css|js|ico|ttf|woff)\.*`
4. Click on the Start button and confirm the dialog box by clicking OK.

⁵²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/post-processors/RegularExpressionExtractorTestPlan.jmx

5. Open the Firefox browser and follow the steps to update the payment information.
6. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
7. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
8. Select all recorded browser actions from WorkBench. Drag and add them as child elements of the thread group.
9. Click on each HTTP request and remove the **Server Name or IP** and **Port Number** options. (We have configured these in HTTP Request Defaults.)
10. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Move this to above the /user/signOut HTTP sampler.
11. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
12. Save the test plan.

Follow these additional steps to configure the *Regular Expression Extractor*.

1. Click on the first HTTP request called /user/addCard and go to Edit ▶ Add ▶ Post Processors. Add Regular Expression Extractor. Configure the options as shown in Figure 5-135.

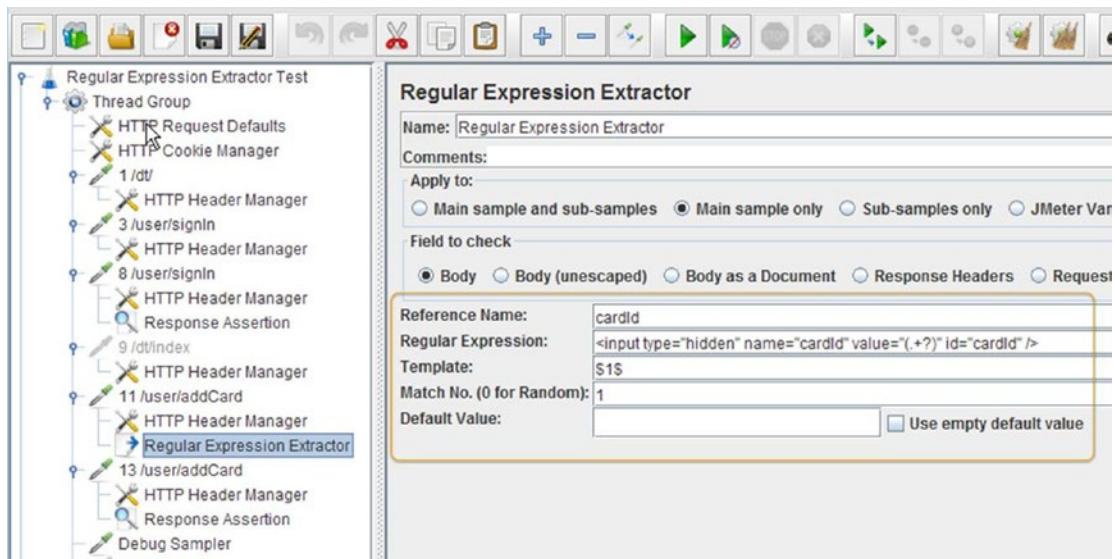


Figure 5-135. Regular expression extractor configuration

2. Click on the second HTTP request called /user/addCard and configure the options as shown in Figure 5-136. (You need to update the card ID, which you are extracting from an earlier request).

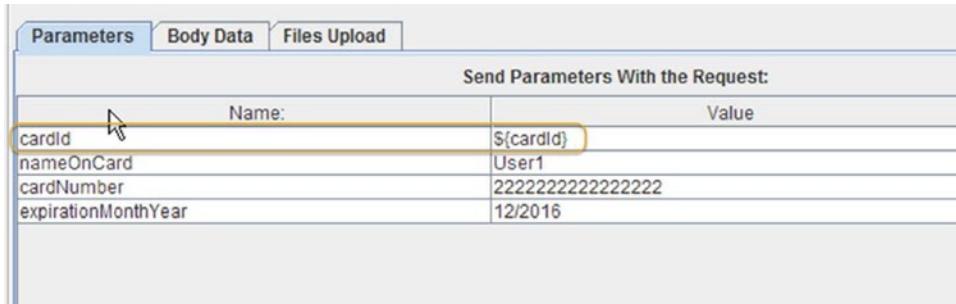


Figure 5-136. HTTP request AddCard

3. Run the test.

The results will be similar to those shown in Figure 5-137.

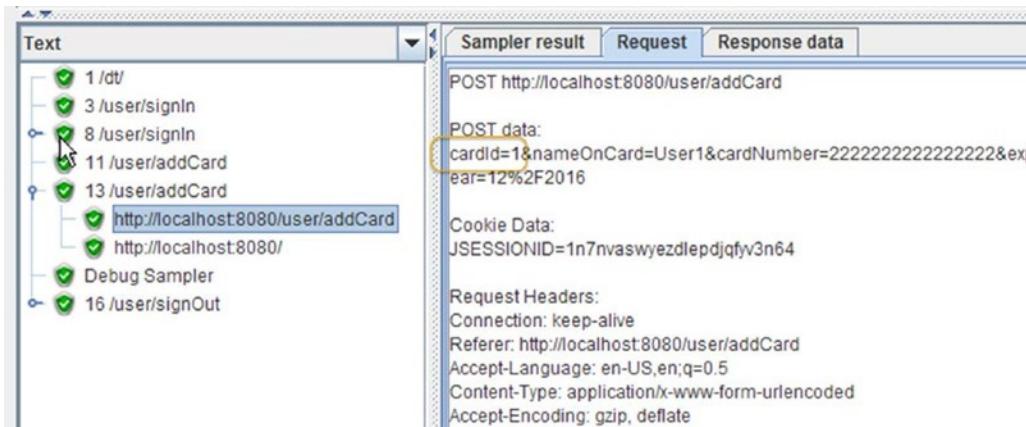


Figure 5-137. Regular expression extractor results

4. You can also click on Debug Sampler to cross-verify the value of cardId after Post-Processor has parsed the value from the sampler response (see Figure 5-138).

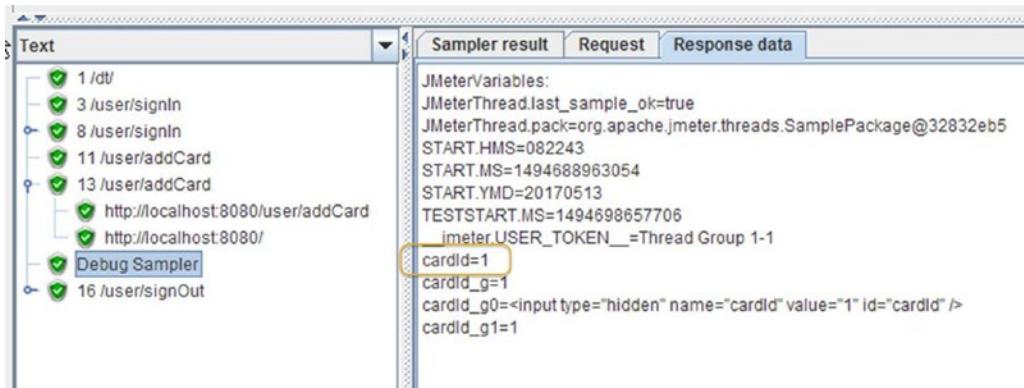


Figure 5-138. Debug sampler

Once the JMeter test is executed, open your browser and launch the Digital Toys Inc. web application. Log in again as user1@dt.com/user1, navigate to the payment information, and verify the updated information.

Properties and Variables

JMeter, like any programming language, has the concept of defining and using name/value pairs. JMeter utilizes this concept for both properties and variables.

Comparison of Properties and Variables

Table 5-4 shows a comparison of the properties and variables.

Table 5-4. Comparison of Properties and Variables

Properties	Variables	User Defined Variables
Defined in jmeter.properties file, JSR223 script, or passed on the command line.	Defined using JSR223 script or CSV Data Config component in a thread group.	Defined using the user defined variable section in the test plan component.
Shared across the thread groups.	Local to the thread group in which it is defined.	Each UDV gets copied as a variable into each of the thread groups at the start of the test execution, after which, it behaves like a variable specific to the thread group.
Use __P() to get the value.	Use \${ } to get the value.	Use \${ } to get the value.

Figure 5-139 shows a big picture view of the properties, variables, and user defined variables.

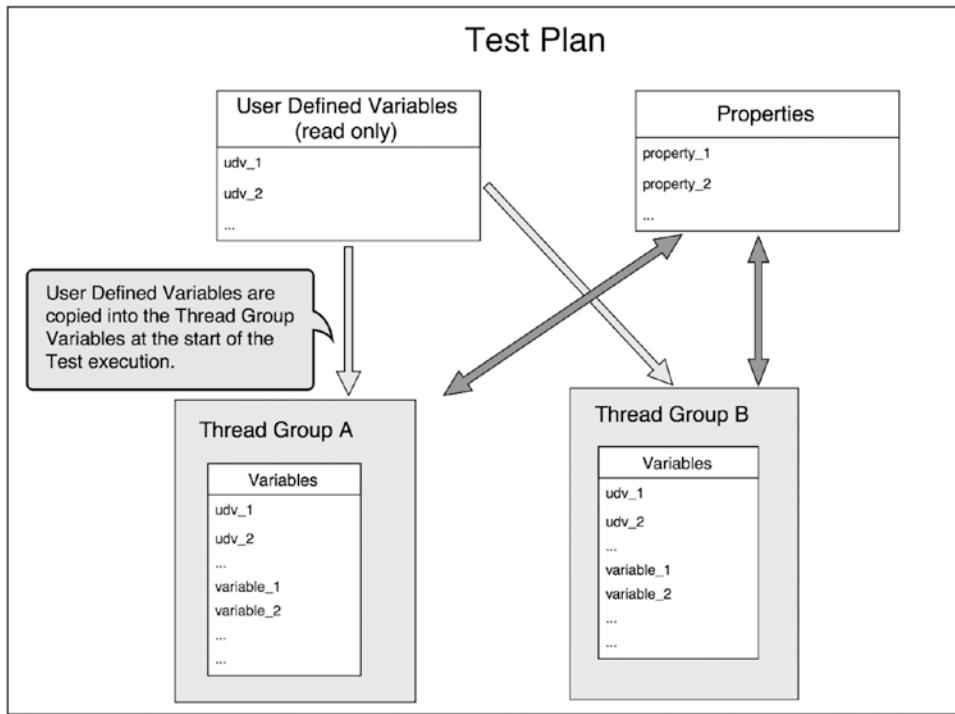


Figure 5-139. Properties, variables, and user defined variables

Notice that the variables are local to a thread group, whereas the modifications to the properties are shared across the thread groups.

Let's illustrate this by the following example.

Follow these steps or download `PropertiesAndVariablesTestPlan.jmx`.⁵³

1. Create a test plan and give it a meaningful name, such as **User Defined Variables Test**. Enable the **Run Thread Groups Consecutively (i.e. Run Groups One at a Time)** checkbox.
2. Click on Test Plan and go to **Edit ▶ Add ▶ Threads (Users)**. Add Thread Group. Configure the name as **Thread Group A** and the **Loop Count** as 1.

⁵³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/properties-and-variables/PropertiesAndVariablesTestPlan.jmx

- Click on Thread Group A and go to Edit ▶ Add ▶ Pre Processor. Add JSR223 Pre Processor. Configure **Script Language** as Java and enter this script: `vars.put("zeta_variable", "variable value");props.put("Alpha_variable", "property value");`. See Figure 5-140.

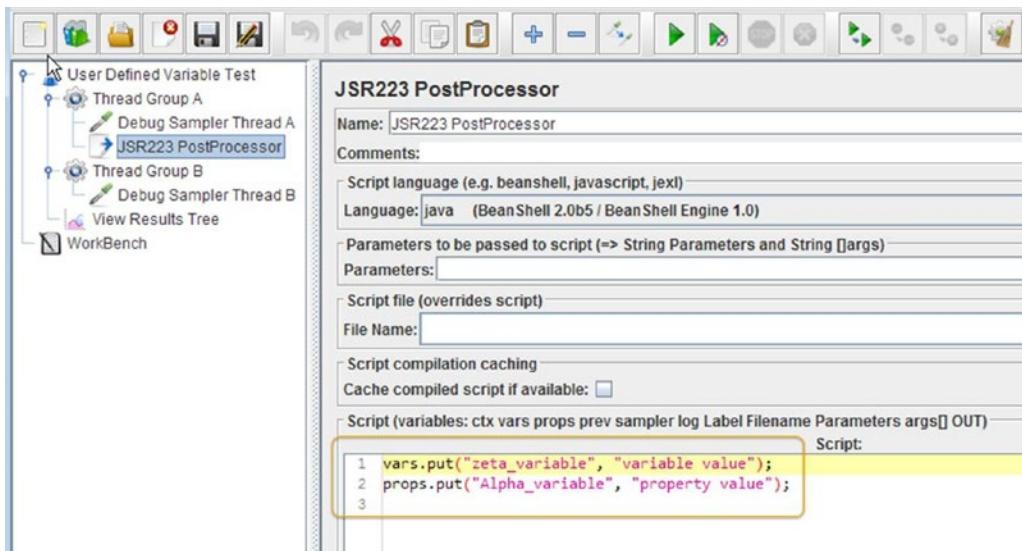


Figure 5-140. JSR223 configuration

- Click on Thread Group A and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Configure **JMeter Properties** as True, **JMeter Variables** as True, and **System Properties** as False (see Figure 5-141).

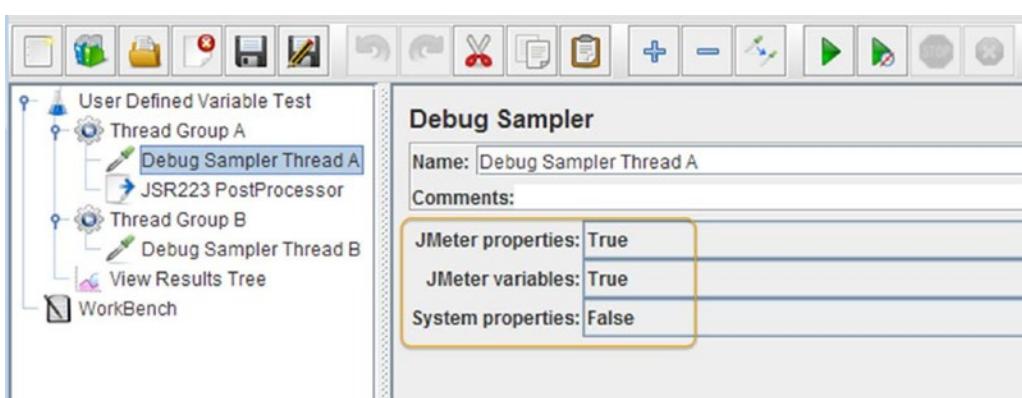
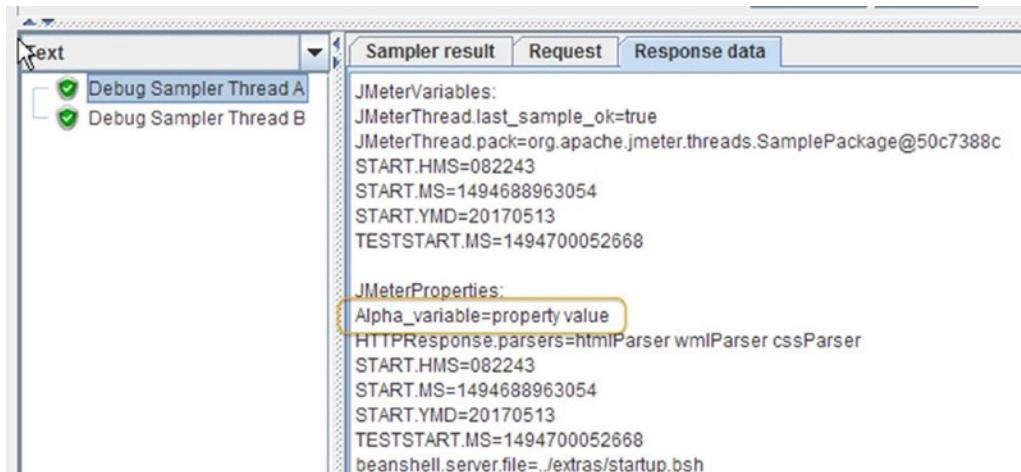


Figure 5-141. Debug sampler configuration

5. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure the name as Thread Group B and the **Loop Count** as 1.
6. Click on Thread Group B and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Configure **JMeter Properties** as True, **JMeter Variables** as True, and **System Properties** as False.
7. Click on Test Plan and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
8. Save the test plan.
9. Run the test.

The results are as shown in Figure 5-142.



The screenshot shows the JMeter interface with a 'Text' listener selected. The left pane displays two thread groups: 'Debug Sampler Thread A' and 'Debug Sampler Thread B'. The right pane shows the 'Sampler result' tab with the following output:

```
JMeterVariables:  
JMETERThread.last_sample_ok=true  
JMETERThread.pack=org.apache.jmeter.threads.SamplePackage@50c7388c  
START.HMS=082243  
START.MS=1494688963054  
START.YMD=20170513  
TESTSTART.MS=1494700052668  
  
JMeterProperties:  
Alpha_variable=property value  
HTTPResponse.parsers=htmlParser wmlParser cssParser  
START.HMS=082243  
START.MS=1494688963054  
START.YMD=20170513  
TESTSTART.MS=1494700052668  
beanshell.server.file=../extras/startup.bsh
```

A yellow box highlights the 'Alpha_variable=property value' entry under 'JMeterProperties'.

Figure 5-142. Results of debug sampler A

Observe that the user parameter `zeta_variable` defined in Thread Group A is not visible in Thread Group B. However, the property `alpha_property` defined in Thread Group A is global and is visible in Thread Group B as well (see Figure 5-143).

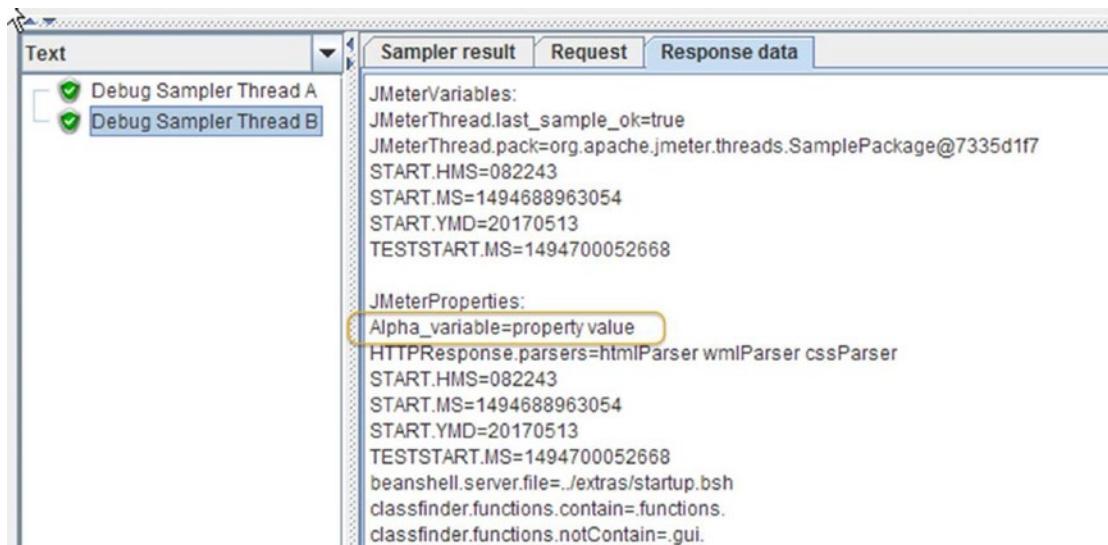


Figure 5-143. Results of debug sampler B

User Defined Variables

The user defined variables (UDV) are copied as variables into every thread group at the start of the test script execution. After that, any changes to the UDV are local to the thread group.

Follow these steps or download UDVEExecuteThreadGroupConsecutivelyTestPlan.jmx.⁵⁴

1. Create a test plan and give it a meaningful name, such as User User Defined Variable Execute Thread Group Consecutively Test. Enable the **Run Thread Groups Consecutively** checkbox. Under **User Defined Variables**, add two **Name:Value** pairs as UDV_Alpha/alpha and UDV_Bravo;bravo (see Figure 5-144).

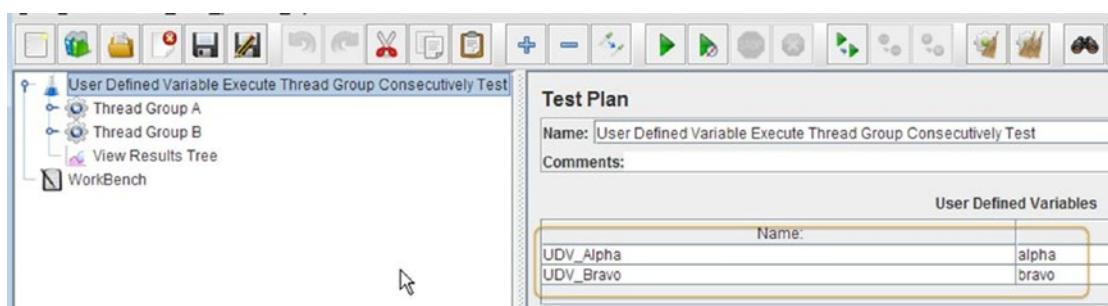


Figure 5-144. User defined variables configuration

⁵⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/properties-and-variables/UDVEExecuteThreadGroupConsecutivelyTestPlan.jmx

2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure the name as Thread Group A and **Loop Count** as 1.
3. Click on Thread Group A and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Configure the name as Debug Sampler Thread A.
4. Click on Thread Group A and go to Edit ▶ Add ▶ Post Processors. Add JSR223 PostProcessor. Configure the language as Java. Add the following in the Script box (see Figure 5-145).

```
log.info("The value of UDV_Bravo before: " + vars.get("UDV_Bravo") );
vars.put("UDV_Bravo", "zulu");
log.info("The value of UDV_Bravo after: " + vars.get("UDV_Bravo") );
```

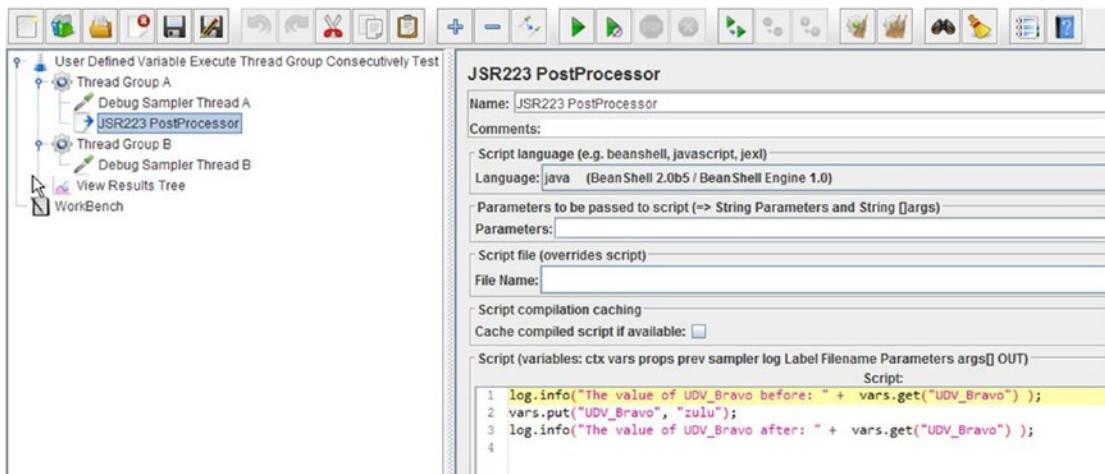


Figure 5-145. JSR223 post-processor configuration

5. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure the name as Thread Group B and the **Loop Count** as 1.
6. Click on Thread Group B and go to Edit ▶ Add ▶ Sampler. Add Debug Sampler. Configure the name as Debug Sampler Thread B.
7. Click on Test Plan and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
8. Save the test plan.
9. Run the test.

Click on the View Results Tree and select Debug Sampler Thread A. You will notice that UDV_Alpha is alpha and UDV_Bravo is bravo, as defined initially in the test plan element (see Figure 5-146).

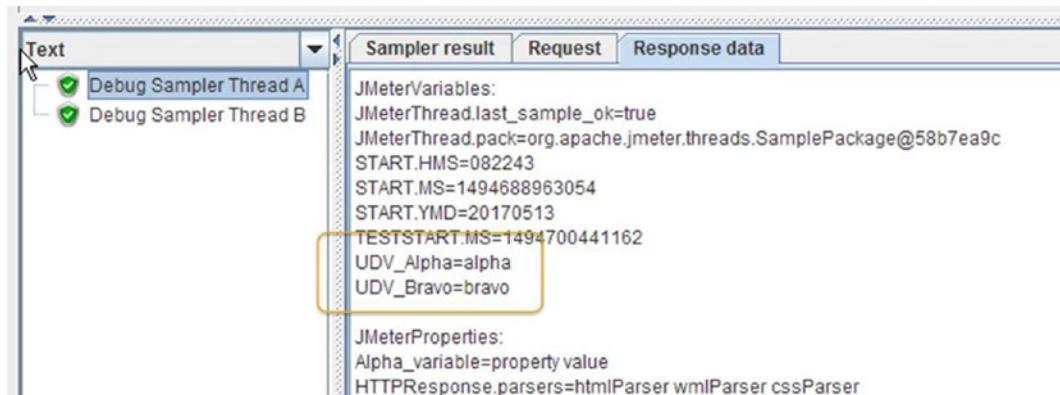


Figure 5-146. Debug sampler A results

The logs show that the value of UDV_Bravo was set to zulu after the JSR223 post-processor executed. However, this change is local to Thread Group A and is not propagated to Thread Group B.

```

2017/05/15 13:33:21 INFO - jmeter.extractor.JSR223PostProcessor: The value of
UDV_Bravo before: bravo
2017/05/15 13:33:21 INFO - jmeter.extractor.JSR223PostProcessor: The value of
UDV_Bravo after: zulu

```

Click on the View Results Tree and select Debug Sampler Thread B. You will find that there was no change to the value of UDV_Bravo (see Figure 5-147).

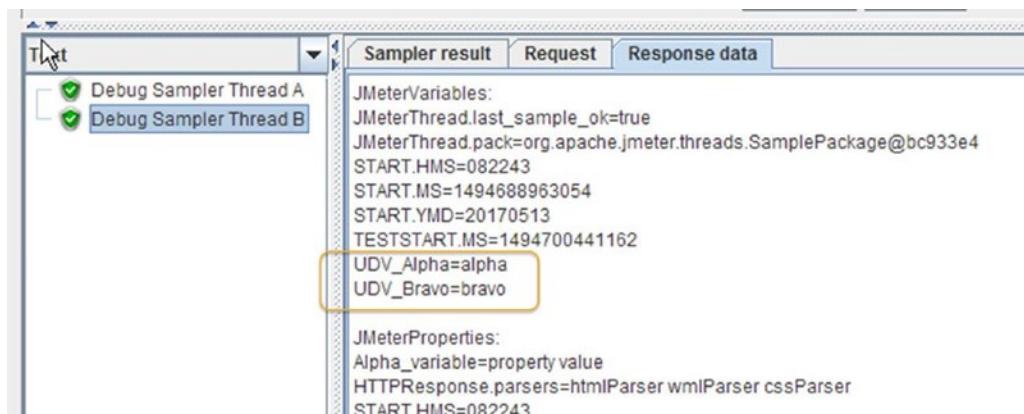


Figure 5-147. Debug sampler B results

Using the Command Line to Initialize Properties

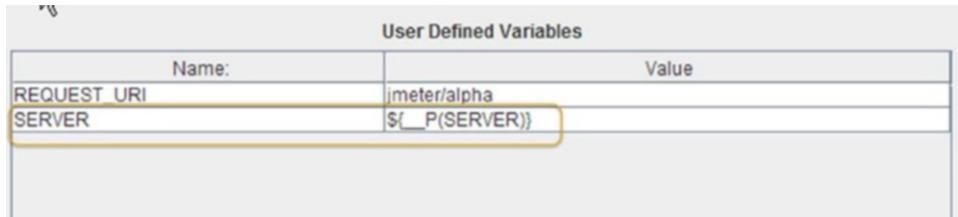
When using non-GUI mode for running the tests, you need to pass different values for various parameters to the test plan based on the environment. For example, the server name is different between the test and the production environments and hence can be defined as a property.

In the following example, the server name has been defined as a property and its value set from the command line using the -J qualifier. A user defined variable has been configured in the test plan element and initialized using the server name property.

Let's illustrate this in the following example.

Follow these steps or download [UDVUsingCmdLineParamTestPlan.jmx](#).⁵⁵

1. Open `UDVExecuteThreadGroupConsecutivelyTestPlan.jmx`.
2. Click on Test Plan and uncheck the **Run Thread Groups Consecutively** checkbox. Add a **Name:Value** pair as `SERVER/${__P(SERVER)}` (see Figure 5-148).



The screenshot shows the 'User Defined Variables' section of the JMeter Test Plan. It contains two entries:

Name:	Value
REQUEST_URI	jmeter/alpha
SERVER	<code>__P(SERVER)</code>

Figure 5-148. Test plan using user defined variables

3. Click on HTTP Request Defaults and update the **Server Name or IP** field to `__P(SERVER)` (see Figure 5-149).

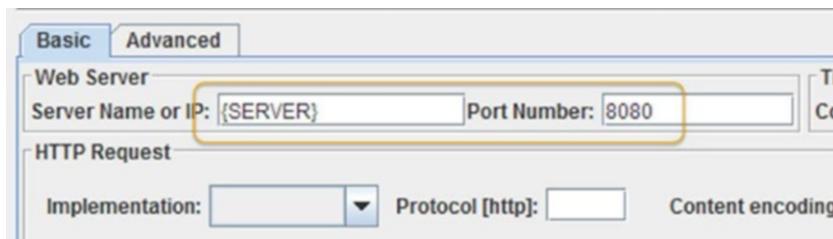


Figure 5-149. Thread group utilizing user defined variable

⁵⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_05/properties-and-variables/UDVUsingCmdLineParamTestPlan.jmx

4. Save the test plan.
5. Run the test from the command line. Issue the following command at the CMD prompt:

```
C:\>jmeter -n -t UDVExecuteThreadGroupConsecutivelyTestPlan.jmx -JSERVER=localhost -l
UDV-cmd-line-test.log
```

```
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using UDVExecuteThreadGroupConsecutivelyTestPlan.jmx
Starting the test @ Mon May 15 14:17:34 PDT 2017 (1494883054405)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 8 in 00:00:02 = 4.2/s Avg: 1135 Min: 1027 Max: 1272 Err: 0 (0.00%)
Tidying up ... @ Mon May 15 14:17:36 PDT 2017 (1494883056495)
... end of run
```

6. Open the test results log file. Issue the following command at the CMD prompt:

```
C:\>cat UDV-cmd-line-test.log
```

```
1494882072124,1232,HTTP Request B,200,OK,Thread Group B 2-2,text,true,,5160,4,8,1230,0
1494882072109,1320,HTTP Request A,200,OK,Thread Group A 1-2,text,true,,5161,4,7,1316,0
1494882072062,1366,HTTP Request B,200,OK,Thread Group B 2-1,text,true,,5161,3,7,1366,0
1494882072359,1090,HTTP Request A,200,OK,Thread Group A 1-3,text,true,,5161,3,5,1086,0
1494882072376,1082,HTTP Request B,200,OK,Thread Group B 2-3,text,true,,5161,2,4,1081,0
1494882072062,1427,HTTP Request A,200,OK,Thread Group A 1-1,text,true,,5162,2,3,1427,0
1494882072609,1035,HTTP Request A,200,OK,Thread Group A 1-4,text,true,,5162,1,2,1035,0
1494882072628,1080,HTTP Request B,200,OK,Thread Group B 2-4,text,true,,5161,1,1,1080,0
```

Note that the server name has been passed from the command line. This was propagated to the server defined under user defined variables. The request samplers referred to this value by enclosing it in \${}.

Similarly, you can configure Number of Threads (Users), Ramp-Up Period (in Seconds), and Loop Count to simulate different loads based on the environment.

Conclusion

In this chapter, you learned to configure your test plan to enable serial and parallel execution of thread groups to simulate real users while performing load testing. You used pre-processors by using the *HTTP URL Rewriting Modifier*. You learned about Logic Controllers for enabling loops based on conditions. You learned about JMeter timers and their appropriate use in test scripts for setting up hard delays to replicate actual user scenario etc., and the *HTTP Request Sampler* to configure real-world HTTP requests. You also learned about the Response Assertion and its various options and how to apply them to responses from samplers and sub-samplers to ensure that tests are running correctly. You also learned about the usage of assertions results, which are especially suited for viewing the results of assertions. Listeners like *View Results Tree*, *View Results In Table*, and *Aggregate Report* Listeners are useful in viewing the results of JMeter test script, saving the test results in CSV and XML formats, and using non-GUI mode to save results to a file. The *Regular Expression Extractor* extracts value from the sampler and stores the value locally for successive samplers. You also learned about the use of JMeter properties, variables, and user defined variables along with their scope with respect to test plans and thread groups.

In the next chapter, you will learn about distributed testing using JMeter in a master-slave environment, various modes of sending results from the slaves to the master, and current limitations of distributed testing.

CHAPTER 6



Distributed Testing

This chapter discusses how to perform distributed testing using JMeter. We cover the prerequisites and configuration of JMeter with remote hosts in master/slave environments. You'll see how to run tests in GUI as well as non-GUI mode and learn about various ways that JMeter sends information from slave(s) to the master. Lastly, there is a section that will be useful in troubleshooting exceptions while you are developing tests script and running in distributed environment.

At the end of this chapter, you'll have a good idea about the distributed load testing approach using JMeter. You will be able to set up a distributed testing environment and trigger test scripts from various JMeter slaves. Those who are already familiar with distributed testing using JMeter can proceed to the next chapter.

When the load generated by JMeter reaches the limit of a client machine in terms of CPU, memory, or network space, then you need to utilize more than one machine. JMeter can be configured to do distributed testing.

In a distributed testing environment, each JMeter client is configured to simulate the load of a few hundred users and the combination of these clients will trigger a few thousand requests simultaneously. It can be thought of as horizontal scaling of load, as you can increase and simulate the test load by adding client machines to the testing environment.

Distributed Testing Using JMeter

Distributed testing is performed using a *master-slave* model. A JMeter *master* node, together with one or more JMeter *slave* nodes, constitute a distributed testing cluster.

The test plan is loaded on the master. The hostname or IP address of the slave machines are configured in the `jmeter.properties` file on the master. This enables those slave machines to be a part of the JMeter distributed testing cluster, and they are visible in the master node GUI. It is assumed that JMeter is already installed on the slave nodes.

The slave nodes obtain a copy of the test plan from the master. The role of the master node is only to orchestrate the test. It is the slave nodes that execute the test and generate the load.

Distributed testing in JMeter is also called *remote testing*.

Prerequisites

The prerequisites for setting up the distributed testing cluster using JMeter are as follows:

- All the slaves must be on the same subnet as the master.
- The application under target should be on the same subnet.
- All slaves and the master should have the same version of JMeter and JVM.

- The firewall on the master and the slaves should be turned off.
- There should be no antivirus software installed on the master or the slaves.
- The network should be stable.
- There should not be any extraneous network activity on the subnet.

Configuration

Open the `jmeter.properties` file on the master and add each slave's hostname or IP address, as shown here:

```
remote_hosts=192.168.0.7,192.168.0.8
```

This is the only configuration that is required.

External configuration files needed by each slave are located on that machine.

For example, in a shopping-cart application where a username and password is required, you could store a list of these in files and load them separately in each slave. Make sure that you have distinct username/password pairs across the entire application to avoid duplicate logins.

On each of the slave machines, create a `users.csv` file, ensuring that the contents are distinct across each machine.

In slave #1, here is the users list.

```
user1@dt.com,user1
user2@dt.com,user2
user3@dt.com,user3
user4@dt.com,user4
user5@dt.com,user5
```

In slave #2, here is the users list.

```
user6@dt.com,user6
user7@dt.com,user7
user8@dt.com,user8
user9@dt.com,user9
user10@dt.com,user10
```

Ensure that the `users.csv` file is placed in the `/bin` folder of the slave's `$JMETER_HOME` directory. When the test executes, it will be picked up.

Let's illustrate this with an example.

Follow these steps or download the `DistributedTestPlan.jmx`¹ file:

1. Create a test plan and give it a meaningful name, such as `Distributed Testing`.
2. Click on Test Plan and go to Edit ▶ Add ▶ *Threads (Users)*. Add Thread Group. Configure **Number of Threads (Users)** as 1 and **Loop Count** as 1.
3. Click on Thread Group and go to Edit ▶ Add ▶ *Config Element*. Add HTTP Cookie Manager.

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_06/DistributedTestPlan.jmx

4. Click on Thread Group and go to Edit ▶ Add ▶ *Config Element*. Add HTTP Request Defaults. Configure **Server Name or IP** as <your_machine_ip_or_hostname> and **Port Name** as 8080.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signIn and **Method** as POST.
6. Click on HTTP Request and go to Edit ▶ Add ▶ *Config Element*. Add CSV Data Set Config and configure as shown in Figure 6-1.

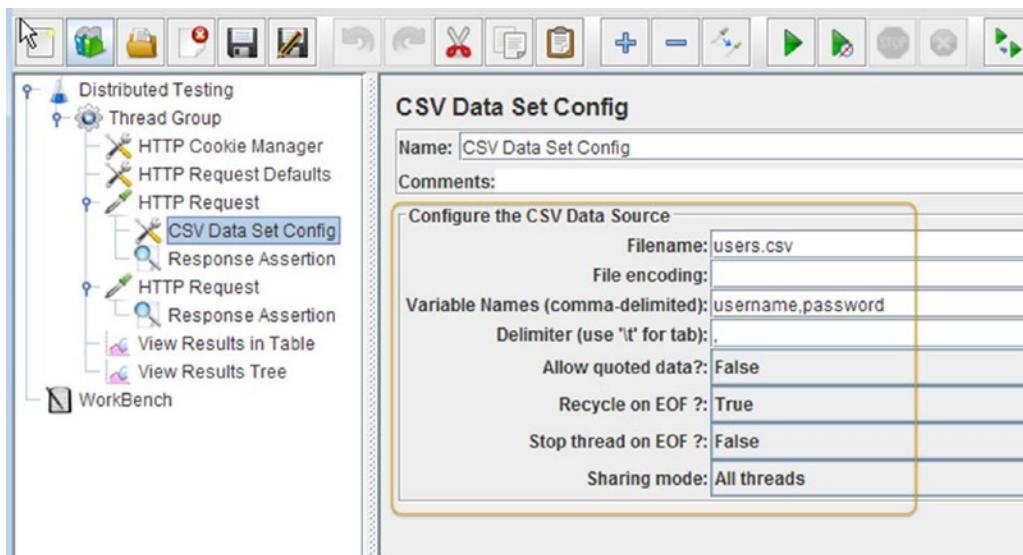


Figure 6-1. CSV data set config

7. Click on HTTP Request again and configure the parameters as shown in Figure 6-2.

	Name:	Value
	email	\${username}
	password	\${password}

Figure 6-2. HTTP request parameters

8. Click on HTTP Request and go to Edit ▶ Add ▶ *Assertions*. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
9. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signOut and **Method** as HEAD.
10. Click on HTTP Request and go to Edit ▶ Add ▶ *Assertions*. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.

11. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
12. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results in Table.
13. Save the test plan.

Running the Test

After updating the `jmeter.properties` file with `remote_hosts`, start the JMeter GUI and load the test plan. Under the Run menu, the following options are available:

- The Remote Start and Remote Start All options will start the test on remote hosts.
- The Remote Stop and Remote Stop All options will stop the test on remote hosts.
- The Remote Exit and Remote Exit All options will stop the JMeter server on remote hosts.
- The Remote Shutdown and Remote Shutdown All options will stop the test on remote hosts but not the JMeter server, and we can continue the test on the remote host by again using the Remote Start option (see Figure 6-3).

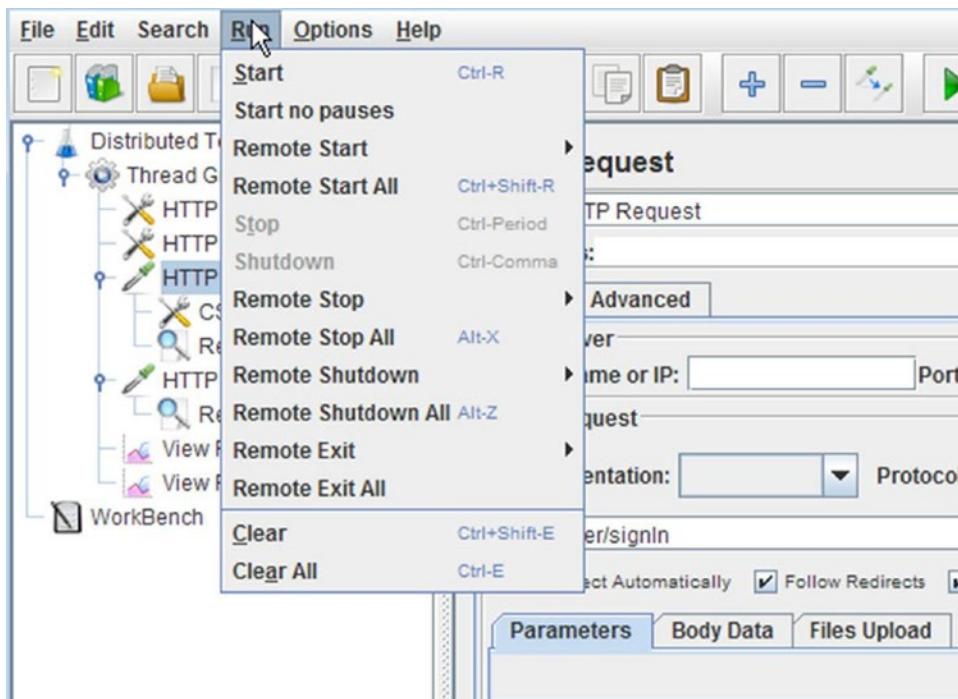


Figure 6-3. JMeter Remote Start option

From the apache-jmeter-3.0/bin directory on each slave, you need to start jmeter-server using the following command:

```
C:\>jmeter-server
```

Once started, it will look like the output shown next.

Remote host #1 jmeter-server logs.

```
C:\> jmeter-server
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter-server.log
Created remote object:
UnicastServerRef [liveRef: [endpoint:[192.168.0.7:51324](local),objID:[535e2c0b:15c0e3a3328:-ffff, 3496195728845345408]]]
```

Remote host #2 jmeter-server logs.

```
C:\> jmeter-server
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter-server.log
Created remote object:
UnicastServerRef [liveRef: [endpoint:[192.168.0.8:63904](local),objID:[-99f93d6:15c0e3a7148:-ffff, -6488513131611751121]]]
```

GUI Mode

You have set up two slaves (remote hosts), which are visible on the JMeter GUI. Run the test on one remote host by selecting Remote Start and selecting any remote host or using the Remote Start All option.

Under Thread Group, you can see that you are running 1 thread and selecting Remote Start All; it will trigger this test on both remote hosts. The *View Results in Table* will show four responses, as shown in Figure 6-4.

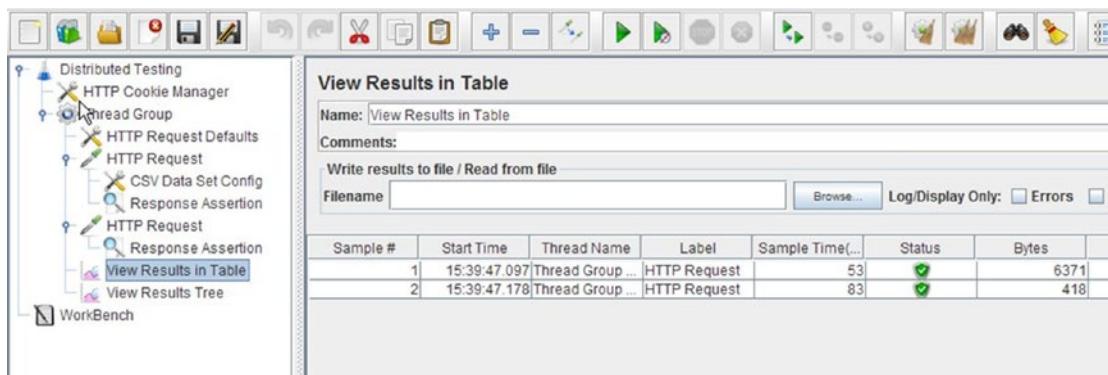


Figure 6-4. Master view results tree

Also on the remote host, the jmeter-server logs will show the test start and the test end.
 Remote host #1 jmeter-server logs.

```
Starting the test on host 192.168.0.7 @ Mon May 15 15:31:15 PDT 2017 (1494887475261)
Finished the test on host 192.168.0.7 @ Mon May 15 15:31:16 PDT 2017 (1494887476397)
```

Remote host #2 jmeter-server logs.

```
Starting the test on host 192.168.0.8 @ Mon May 15 15:31:15 PDT 2017 (1494887475062)
Finished the test on host 192.168.0.8 @ Mon May 15 15:31:15 PDT 2017 (1494887475390)
```

Non-GUI Mode

GUI mode takes a lot of memory. JMeter provides an option to do a *remote run* of the tests.

Execute this command with -R and add the remote hosts' IP addresses.

```
C:\>jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7,192.168.0.8
```

It will show the following output. You can see that tests are executed on remote engines.

```
C:\>jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7,192.168.0.8
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using DistributedTestPlan.jmx
Configuring remote engine: 192.168.0.7
Configuring remote engine: 192.168.0.8
Starting remote engines
Starting the test @ Mon May 15 15:36:44 PDT 2017 (1494887804089)
summary = 0 in 00:00:00 = *****/s Avg: 0 Min: 9223372036854775807 Max:
-9223372036854775808 Err: 0 (0.00%)
```

```
Tidying up remote @ Mon May 15 15:36:45 PDT 2017 (1494887805458)
```

```
Remote engines have been started
```

```
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 2 in 00:00:01 = 3.7/s Avg: 106 Min: 105 Max: 107 Err: 0 (0.00%)
```

```
Tidying up remote @ Mon May 15 15:36:45 PDT 2017 (1494887805850)
```

```
... end of run
```

```
... end of run
```

Execute the following command with -r without the remote hosts' IP addresses. This command will take the remote hosts from the `jmeter.properties` file assigned to the `remote_hosts` property.

```
C:\>jmeter -n -t DistributedTestPlan.jmx -r
```

RMI Port

By default, the `server_port` is set to 1099. Sometimes it may be that this port is blocked, and we are not able to start JMeter in the *master-slave* environment. In this case, we need to set the `server_port` of the slaves to something else.

Open the `jmeter.properties` file on the slaves and change it to a different port number, such as 1234.

```
# RMI port to be used by the server (must start rmiregistry with same port) server_port=1234
```

The command for running tests should be as follows:

```
C:\> jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7:1234, 192.168.0.8:1234
```

If you forget to use the updated port number in the command, you will get the exception shown here:

```
C:\> jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7,192.168.0.8:1234
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using DistributedTestPlan.jmx
Configuring remote engine: 192.168.0.7:1234
Connection refused to host: 192.168.0.7; nested exception is:
    java.net.ConnectException: Connection refused: connect
Failed to configure 192.168.0.7:1234
Configuring remote engine: 192.168.0.8:1234
Connection refused to host: 192.168.0.8; nested exception is:
    java.net.ConnectException: Connection timed out: connect
Failed to configure 192.168.0.8:1234
Stopping remote engines
Remote engines have been stopped
Error in NonGUIDriver java.lang.RuntimeException: Following remote engines could not be
configured:[192.168.0.7:1234,
192.168.0.8:1234]
```

Sample Sender Mode

In distributed testing of the master-slave environment, remote hosts (slaves) do the test execution and send samples to the client (master). The master gathers input from all the remote hosts and consolidates into a single view with respect to the target application server.

In the overall process, based on the configuration, remote hosts coordinate with the master to send samples before executing the next thread. This affects the maximum throughput of the server test, as the sample result has to be sent back before the thread can continue.

The client node (master) needs to have one of the following sample sending modes:

- **Standard:** Send samples synchronously as soon as they are generated.
`#mode=Standard`
- **Batch Mode:** Send saved samples when either the count (`num_sample_threshold`) or time (`time_threshold`) exceeds a threshold, at which point the samples are sent synchronously. The thresholds can be configured on the server using the following properties:

`num_sample_threshold`: The number of samples to accumulate; default is 100

```
time_threshold: The time threshold; default is 60000 ms = 60 seconds
```

```
#mode=Batch
...
#num_sample_threshold=100
# Value is in milliseconds
#time_threshold=60000
...
```

- **Statistical Mode:** Send a summary sample when either the count or time exceeds a threshold. The samples are summarized by thread group name and sample label.

The following fields are accumulated:

- Elapsed time
- Latency
- Bytes
- Sample count
- Error count

Other fields that vary between samples are lost.

```
#mode=Statistical
#Set to true to key statistical samples on threadName rather than threadGroup
#key_on_threadname=false
...
#num_sample_threshold=100
# Value is in milliseconds
#time_threshold=60000
...
```

- **Hold Mode:** Hold samples in an array until the end of a run. This may use a lot of memory on the server and is discouraged.

```
#mode=Hold
```

- **DiskStore Mode:** Store samples in a disk file (under java.io.temp) until the end of a run. The serialized data file is deleted on JVM exit.

```
# DiskStore: as for Hold mode, but serialises the samples to disk, rather than
saving in memory
#mode=DiskStore
```

- **StrippedDiskStore Mode:** Remove responseData from successful samples and use DiskStore sender to send them.

```
# Same as DiskStore but strips response data from SampleResult
#mode=StrippedDiskStore
```

- **Stripped Mode:** Remove responseData from successful samples.

```
#mode=Stripped
```

- **StrippedBatch Mode:** Remove responseData from successful samples and use Batch sender to send them.

```
#mode=StrippedBatch
```

- **Asynch:** Samples are temporarily stored in a local queue. A separate worker thread sends the samples. This allows the test thread to continue without waiting for the result to be sent back to the client (the master). However, if samples are being created faster than they can be sent, the queue will eventually fill up, and the sampler thread will block until some samples can be drained from the queue. This mode is useful for smoothing out peaks in sample generation. The queue size can be adjusted by setting the JMeter property `asynch.batch.queue.size` (default 100) on the server node.

```
# Asynchronous sender; uses a queue and background worker process to return the
samples #mode=Asynch
# default queue size
#asynch.batch.queue.size=100
```

- **StrippedAsynch Mode:** Remove responseData from successful samples and use Async sender to send them.

```
# Same as Async but strips response data from SampleResult
#mode=StrippedAsynch
```

- **Custom Implementation Mode:** Set the mode parameter to custom sample sender class name. This must implement the interface `SampleSender` and have a constructor that takes a single parameter of type `RemoteSampleListener`.

```
#mode=org.example.load.MySampleSender
```

Open the `jmeter.properties` file and set the mode as per the requirements.

Note Stripped modes (`StrippedDiskStore`, `Stripped`, `StrippedBatch`, and `StrippedAsynch`) strip `responseData`, meaning that some elements that rely on the previous `responseData` being available will not work. Give attention to this feature while developing the test script.

Unreachable Remote Hosts

The Unreachable Remote Host condition will happen when one or more of the remote hosts is not reachable by the client (the master). Perhaps they have not yet booted up or they are shut down. In this case, when you trigger the test, it will fail.

```
C:\>jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7:1234,192.168.0.8:1234
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using DistributedTestPlan.jmx
Configuring remote engine: 192.168.0.7:1234
Connection refused to host: 192.168.0.7; nested exception is:
    java.net.ConnectException: Connection refused: connect
Failed to configure 192.168.0.7:1234
Configuring remote engine: 192.168.0.8:1234
Connection refused to host: 192.168.0.8; nested exception is:
    java.net.ConnectException: Connection timed out: connect
Failed to configure 192.168.0.8:1234
Stopping remote engines
Remote engines have been stopped
Error in NonGUIDriver java.lang.RuntimeException: Following remote engines could not be
configured:[192.168.0.7:1234,
192.168.0.8:1234]
```

In the first case, when the remote hosts are still booting up, JMeter has a property to wait for some time and then trigger the test. You can set how many retries JMeter has to make and how much retry delay it has to wait before starting the test.

By default, `client.tries` is set to 1 and `client.retries_delay` is set to 5000 milliseconds.
Uncomment these properties and rerun the test.

```
# When distributed test is starting, there may be several attempts to initialize
# remote engines. By default, only a single try is made. Increase the following property
# to make it retry additional times
client.tries=1

# If there are initialization retries, the following property sets a delay between attempts
client.retries_delay=5000
```

JMeter has a property to skip the remote hosts if they are not reachable and continue the test with the rest.
Update `client.continue_on_fail` to true under the `jmeter.properties` file.

```
# When all initialization tries are made, test will fail if some remote engines are failed
# Set the following property to true to ignore failed nodes and proceed with test
client.continue_on_fail=true
```

Run the test again and it will skip the unreachable host.

```
C:\>jmeter -n -t DistributedTestPlan.jmx -R 192.168.0.7,192.168.0.8
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using DistributedTestPlan.jmx
```

```

Configuring remote engine: 192.168.0.7
Configuring remote engine: 192.168.0.8
Connection refused to host: 192.168.0.8; nested exception is:
    java.net.ConnectException: Connection timed out: connect
Failed to configure 192.168.0.8
Following remote engines could not be configured:[192.168.0.8]
Continuing without failed engines...
Starting remote engines
Starting the test @ Mon May 15 17:04:14 PDT 2017 (1494893054551)
Remote engines have been started
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 2 in 00:00:01 = 2.2/s Avg: 376 Min: 279 Max: 474 Err: 0 (0.00%)
Tidying up remote @ Mon May 15 17:04:16 PDT 2017 (1494893056800)
... end of run

```

Limitations

The limitations with JMeter in distributed testing are listed here:

- It is quite expensive to set up dedicated hardware for performance testing on the premises. A cloud-based distributed testing environment will provide a solution to this limitation.
- RMI cannot communicate across subnets without a proxy; therefore, neither can JMeter.
- Since JMeter sends all the test results to the controlling console, it is easy to saturate the network. It is a good idea to use the `Simple Data Writer` to save the results and view the file later with one of the graph Listeners.
- A single JMeter client running on a 2-3 GHz CPU can handle 300-600 threads depending on the type of test. (The exception is the web-services). XML processing is CPU intensive and will rapidly consume all the CPU cycles. As a general rule, performance of XML-centric applications is 4-10 times slower than applications using binary protocols.

Conclusion

In this chapter, you learned to distribute load generation by using multiple machines, configuring remote hosts, and verifying that the remote hosts have successfully run the test. You also learned about the limitations of distributed testing using JMeter. In the next chapter, you will learn JMeter best practices that will make you a more efficient user of JMeter.

CHAPTER 7



JMeter Best Practices

This chapter has examples, each illustrating a concept, feature, or a best practice of JMeter.

At the end of this chapter, you'll have more insight into JMeter and understand little efficient techniques that will help you write better and faster tests. It is not mandatory to read through this chapter, and if you want to skip to next chapter, you can do so.

HTTP Request Defaults

Always use the *HTTP Request Defaults* configuration element in a test plan. The most important elements are the Server Name or IP and Port Number.

This is the place to configure the proxy server information. This is also the place to store any REST API key/values as request parameters.

To make the test plan more portable, you should use *HTTP Request Defaults* and populate the Server Name or IP and Port Number fields with the server name and port information relevant to the test environment. You should use relative URLs in the HTTP Requests field. This is a great advantage, as just by changing the server and port number fields, you can re-target the test plan for multiple environments.

For example, say you have production servers hosted at <http://www.xyz.com>, company servers hosted at `test.xyz.com`, and a developer environment in the intra-net at a private IP address. In such a scenario, you can reuse the test plan by merely changing the Server Name or IP and Port Number options inside *HTTP Request Defaults* (see Figure 7-1).

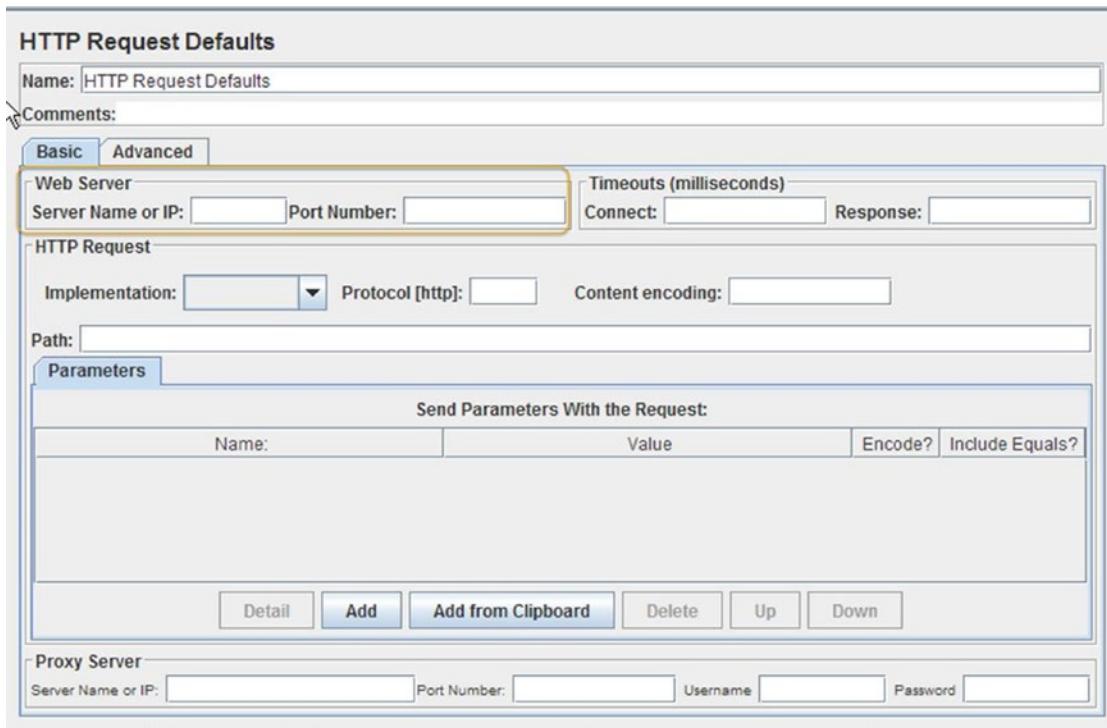


Figure 7-1. HTTP defaults

Follow Redirects

Always enable the Follow Redirects option.

When you browse web sites or are using a web application, sometimes the browser redirects to a new URL using the concept of URL redirection. This is also called URL forwarding. HTTP status codes 3XX are used to indicate redirection.

When you add an *HTTP Request Sampler* to the test plan, the Follow Redirects flag is enabled by default. You will see what happens when this flag is not checked in the following example.

Follow these steps or download `FollowRedirectTestPlan.jmx`.¹

1. Create a test plan and give it a meaningful name, such as **Follow Redirect Test**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Sampler**. Add **HTTP Request**. Configure **Server Name or IP** as **localhost**, **Port Number** as **8080**, and **Path** as **/dt/info**. Uncheck the **Follow Redirects** checkbox.
4. Click on **HTTP Request** and go to **Edit > Add > Assertions**. Add **Response Assertion**. Configure **Response Field to Test** as **Response Code**, **Pattern Matching Rules** as **Equals**, and **Patterns To Test** as **200**.

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_07/FollowRedirectTestPlan.jmx

5. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree (see Figure 7-2).

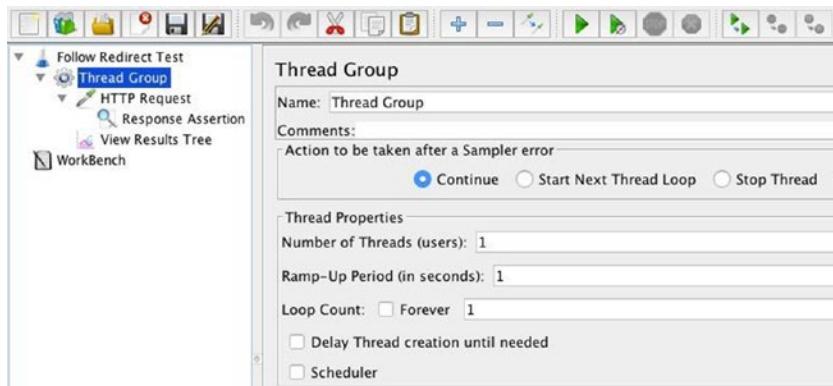


Figure 7-2. Follow redirect test

6. Save the test plan.
7. Run the test.

The results will be similar to those shown in Figure 7-3.

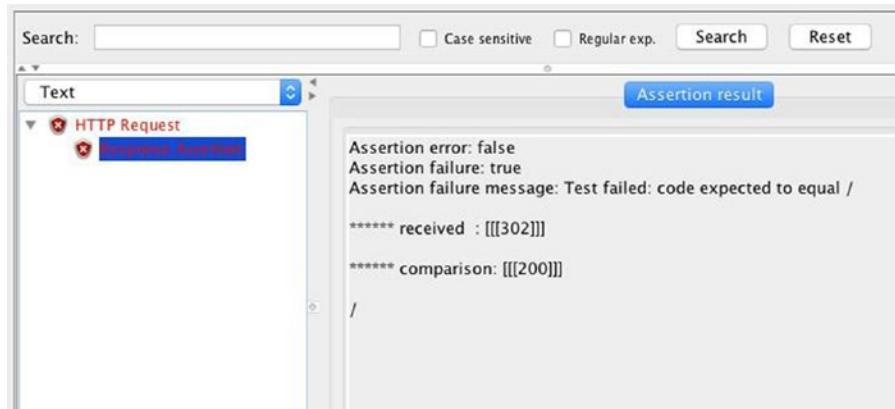


Figure 7-3. Assertion failures

Observe the sampler results shown in Figure 7-4. The assertion failed as the response code was 302 instead of 200. Also note that the Location field in the Response Headers indicates the correct URL to be `http://localhost:8080/dt/newInfo`.

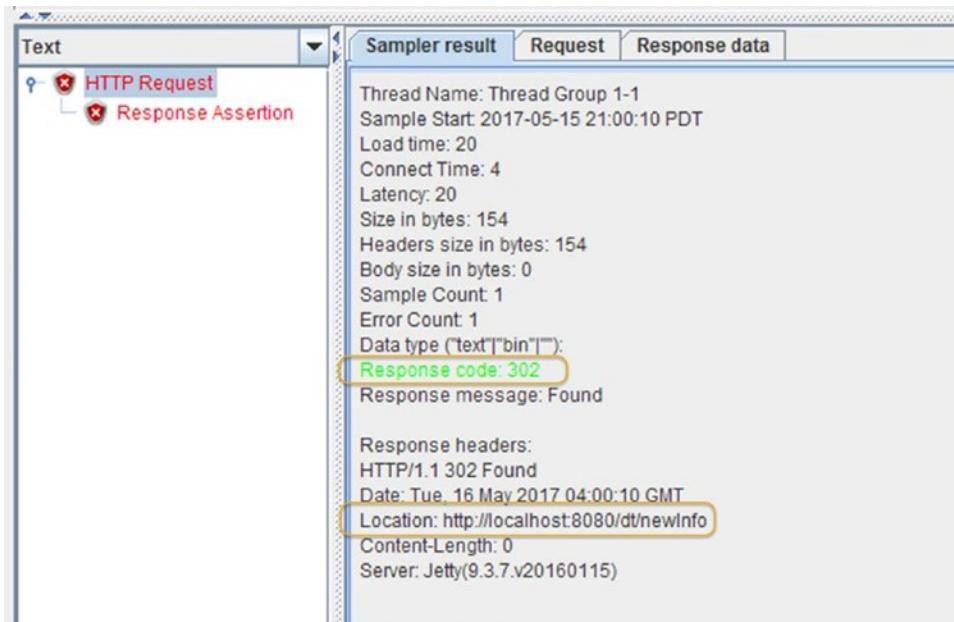


Figure 7-4. Sampler results

Of course, the test will succeed when you select the Follow Redirects flag in the *HTTP Request Sampler*.

Note It is a best practice to select the Follow Redirects flag. Also, it is equally important to add assertions during test script development to make sure that we detect the failures.

Cookie Manager

Always use a *cookie manager*, as cookies are the most common mechanism for web applications to maintain session state.

Consider the following use-case for placing an order from the Digital Toys Inc. web application.

1. Sign in (user user1@dt.com, password user1).
2. Check Detail.
3. Choose Add To Cart.
4. Check out.
5. Add the billing/shipping address.
6. Add the credit card details.

7. Place your order.
8. View the order history.
9. Sign out.

One of the best features of JMeter is the *HTTP(S) Test Script Recorder*. Let's use it to record this use-case by following these steps or downloading *CookieManagerTestPlan.jmx*.²

1. Create a test plan and give it a meaningful name, such as *Cookie Manager Test*.
2. Click on WorkBench and go to Edit ▶ Add ▶ Non-Test Elements. Add *HTTP(S) Test Script Recorder*. Choose Global Settings and then configure **Port** as 7070, **Test Plan Content, Target Controller** as *WorkBench > HTTP(S) Test Script Recorder*, and **URL Patterns to Exclude** as *.*\.(css|js|ico|ttf|woff).**
3. Click on the Start button.
4. Use the browser and follow the use-case steps to place an order on the Digital Toys web application.
5. Click on test plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
6. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
7. Save the test plan.
8. Run the test.
9. Log in to the Digital Toys Inc. web application and verify the order (see Figure 7-5).

S. No.	Product Name	Price
1.	Good Shot Camera	\$ 125.95
	Total	\$ 125.95

Figure 7-5. Order History 1

²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_07/CookieManagerTestPlan.jmx

The process of recording the use-case created an order. After that, the JMeter test execution should have created another order. However, looking under Order History, we only find one order. Why is the second order missing? The JMeter test execution was not successful in placing an order because the session was not maintained between requests. Let's fix this by adding an *HTTP Cookie Manager*.

1. Now click on test plan and go to **Edit > Add > Config Element**. Add **HTTP Cookie Manager**.
2. Run the test.
3. Log in to the Digital Toys Inc. web application and verify the order (see Figure 7-6).

The screenshot shows the 'Your Order History' section of the Digital Toys Inc. website. It displays two distinct order entries:

- Order Date: 2016-05-30**
Status: Order Placed and Shipped

S. No.	Product Name	Price
1.	Good Shot Camera	\$ 125.95
	Total	\$ 125.95

- Order Date: 2016-05-30**
Status: Order Placed and Shipped

S. No.	Product Name	Price
1.	Good Shot Camera	\$ 125.95
	Total	\$ 125.95

A blue 'Continue Shopping' button is located at the bottom right of the main content area. A copyright notice at the bottom states: Copyright© 2016 - JMeter By Example - All rights reserved.

Figure 7-6. Order history 2

4. Go to Order History under the Digital Toys Inc. web application and verify the recently placed order. You will see that there were two orders placed.

Note It is a best practice to always use an *HTTP Cookie Manager*. It should be a child element of Thread Group, as this will ensure that each thread is uniquely identified by the server.

Cache Manager

The Cache Manager simulates a browser cache. You can make JMeter behave closer to a real browser by including an *HTTP Cache Manager*.

JMeter is not a real browser. It does not execute JavaScript or render content. A typical web page has many media files, such as images, video, audio, and script files. A good web page design includes headers that tell the proxy servers and browsers to cache these resources so that they are downloaded from the origin the first time and later fetched from the cache. The *HTTP Cache Manager*, if included, will cache the cache-able resources.

Each thread has its own cache and you can specify the limit on the cache size by configuring the Max Number of Resources to Cache field. Setting this value too high will increase the memory required by JMeter (see Figure 7-7).

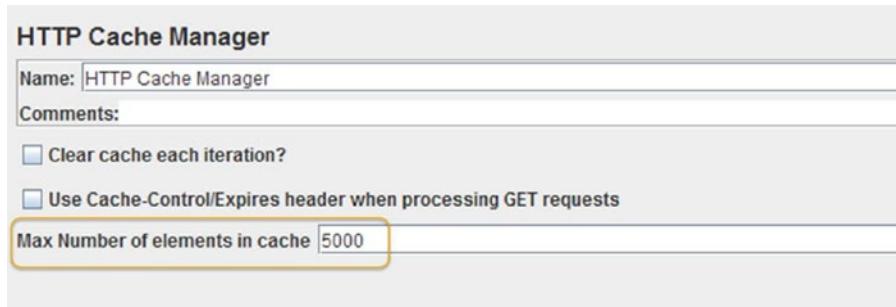


Figure 7-7. Cache manager

JMeter Using Maven

Apache Maven is a popular build-management tool. It provides a standard method to build the software executable and various other project artifacts. It provides a plugin mechanism to support additional tasks.

Inside pom.xml, add the following XML code snippet.³

```
<build>
    <plugins>
        <plugin>
            <groupId>com.lazerycode.jmeter</groupId>
            <artifactId>jmeter-maven-plugin</artifactId>
            <version>1.10.1</version>
            <executions>
                <execution>
                    <id>jmeter-tests</id>
                    <phase>verify</phase>
                    <goals>
                        <goal>jmeter</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

³https://github.com/Apress/pro-apache-jmeter/tree/master/Matam_Ch_07/jmeter-mvn-example

```

        </execution>
    </executions>
</plugin>
</plugins>
</build>
```

Issue the following command in the CMD prompt to run the JMeter test.

```
c:\>mvn verify
```

Output will be similar to the following:

```

[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building JMeter Maven Example 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ jmeter-mvn-example ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build
is platform dependent!
[INFO] skip non existing resourceDirectory c:\github\jmeter-mvn-example\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ jmeter-mvn-example ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ jmeter-mvn-
example ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build
is platform dependent!
[INFO] skip non existing resourceDirectory c:\github\jmeter-mvn-example\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ jmeter-mvn-example ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ jmeter-mvn-example ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ jmeter-mvn-example ---
[WARNING] JAR will be empty - no content was marked for inclusion!
[INFO]
[INFO] --- jmeter-maven-plugin:1.10.1:jmeter (jmeter-tests) @ jmeter-mvn-example ---
[INFO]
[INFO] -----
[INFO] P E R F O R M A N C E      T E S T S
[INFO] -----
[INFO]
[INFO]
[info]
```

```
[debug] JMeter is called with the following command line arguments: -n -t c:\github\jmeter-mvn-example\src\test\jmeter\GoogleSearch.jmx -l c:\github\jmeter-mvn-example\target\jmeter\results\20170515-GoogleSearch.jtl -d c:\github\jmeter-mvn-example\target\jmeter\logs\GoogleSearch.jmx.log
[info] Executing test: GoogleSearch.jmx
[debug] Creating summariser <summary>
[debug] Created the tree successfully using c:\github\jmeter-mvn-example\src\test\jmeter\GoogleSearch.jmx
[debug] Starting the test @ Mon May 15 22:17:42 PDT 2017 (1494911862824)
[debug] Waiting for possible shutdown message on port 4445
[debug] summary = 1 in 1.2s = 0.8/s Avg: 440 Min: 440 Max: 440 Err: 0 (0.00%)
[debug] Tidying up ... @ Mon May 15 22:17:44 PDT 2017 (1494911864099)
[debug] ... end of run
[info] Completed Test: GoogleSearch.jmx
[INFO]
[INFO] Test Results:
[INFO]
[INFO] Tests Run: 1, Failures: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.905 s
[INFO] Finished at: 2017-05-15T22:17:44-07:00
[INFO] Final Memory: 18M/223M
[INFO] -----
```

Passing Variables Across Thread Groups

Sometimes, there is a need to pass information from one thread group to another. We can achieve this by setting the JMeter Property value in one thread group and using its value in the other thread group.

Let's assume that we need to extract the URL from the response of an *HTTP Request* located in Thread Group A and use this value to update the server name or IP of *HTTP Request* belonging to the subsequent Thread Group B.

Follow these steps or download `PassingVariableTestPlan.jmx`.⁴

1. Create a test plan and give it a meaningful name, such as **Passing Variable Test**. Choose Test Plan and then check the **Run Thread Groups Consecutively** (i.e. run groups one at a time) property (see Figure 7-8).

⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_07/PassingVariableTestPlan.jmx

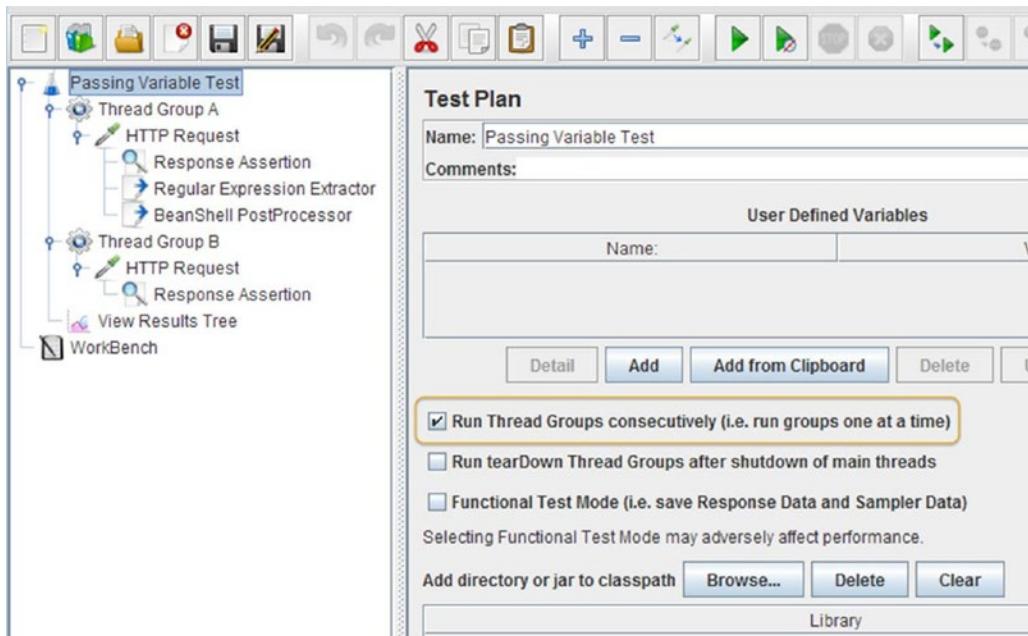


Figure 7-8. Run thread groups consecutively

2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure the name as Thread Group A.
3. Click on Thread Group A and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /.
4. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Click on the HTTP Request of Thread Group A and go to Edit ▶ Add ▶ Post Processor. Add Regular Expression Extractor. Configure **Reference Name** as url1, **Regular Expression** as http://(.*)8080/, **Template** as \$1\$, **Match No.(0 for Random)** as 1, and **Default Value** as NONE (see Figure 7-9).

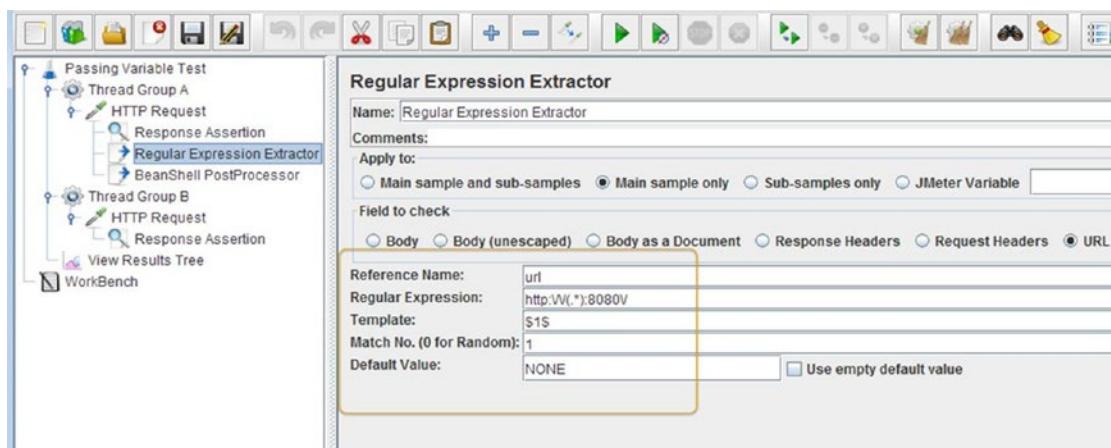


Figure 7-9. Regular expression extractor

6. Click on *HTTP Request* of Thread Group A and go to Edit ▶ Add ▶ Post Processor. Add BeanShell PostProcessor. Add this code to the **Script** text area: \${__setProperty(url, \${url})}. See Figure 7-10.

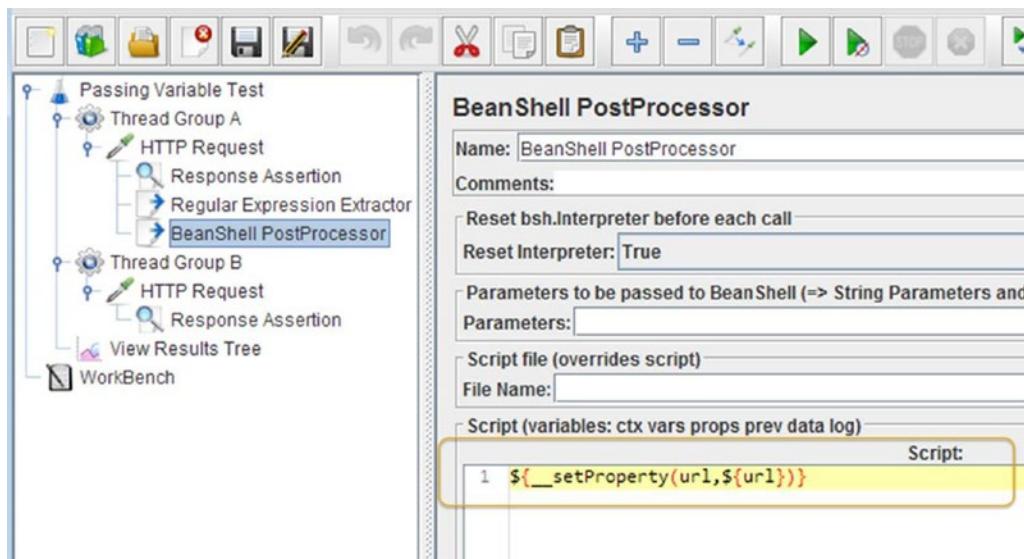


Figure 7-10. BeanShell PostProcessor

7. Click again on the test plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure the name as Thread Group B.
8. Click on Thread Group B and go to Edit ▶ Add ▶ Sampler and add HTTP Request. Configure **Server Name or IP** as \${__property(url)}, **Port Number** as 8080, and **Path** as / (see Figure 7-11).

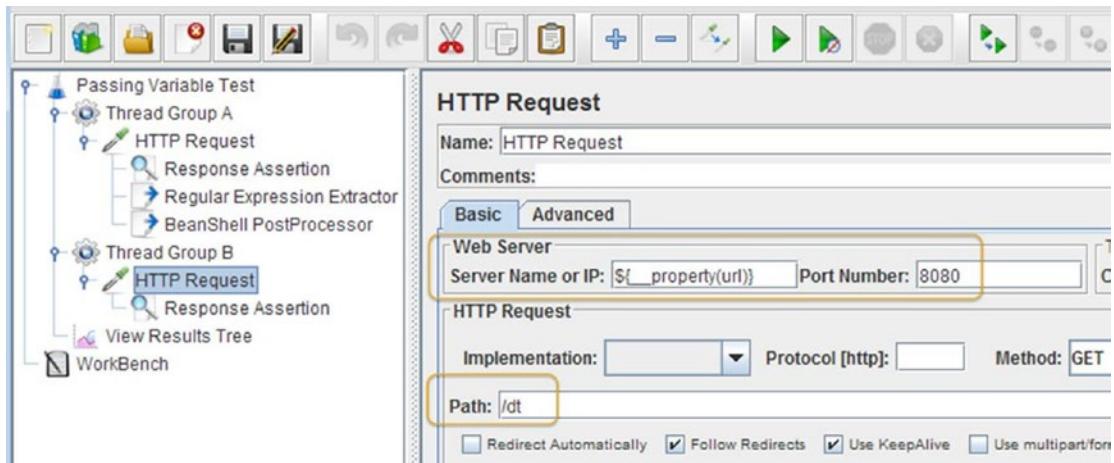


Figure 7-11. Variable from first thread group

9. Click on HTTP Request and go to Edit > Add > Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
10. Click on test plan and go to Edit > Add > Listener. Add View Results Tree.
11. Save the test plan.
12. Run the test.

Results are shown in Figure 7-12.

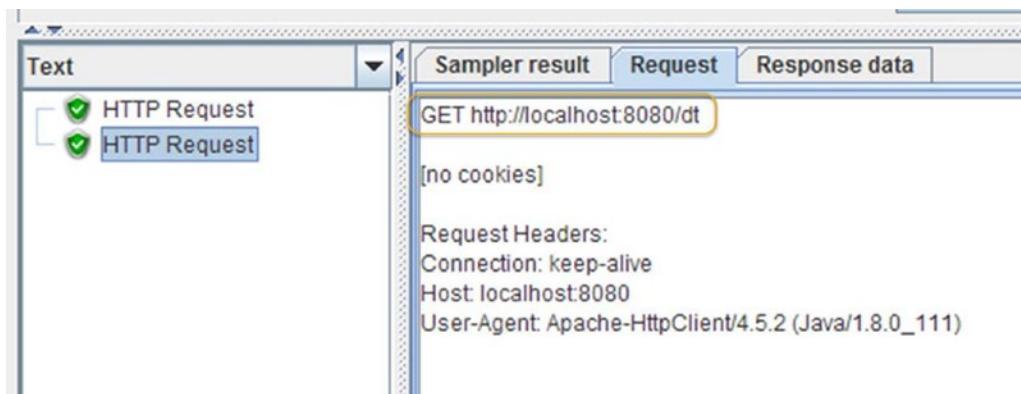


Figure 7-12. Variable substitution

The *regular expression extractor* extracts the URL and sets the URL variable, which is used by the *BeanShell PostProcessor* to define an URL property. The URL property is accessible to the Thread Group B and is de-referenced in the server or IP parameter of the HTTP request. You can see that URL variable has been replaced by the actual value in the second Thread Group B.

Running Parallel Thread Groups

In the real world, many users are performing different use-cases on the web application. To mimic this behavior, JMeter has a way to run thread groups in parallel.

Follow these steps or download [ParallelThreadGroupTestPlan.jmx](#).⁵

1. Create a test plan and give it a meaningful name, such as **Parallel Thread Group Test**. Configure Test Plan and uncheck the **Run Thread Groups Consecutively** (i.e. run groups one at a time) property if it's checked (see Figure 7-13).

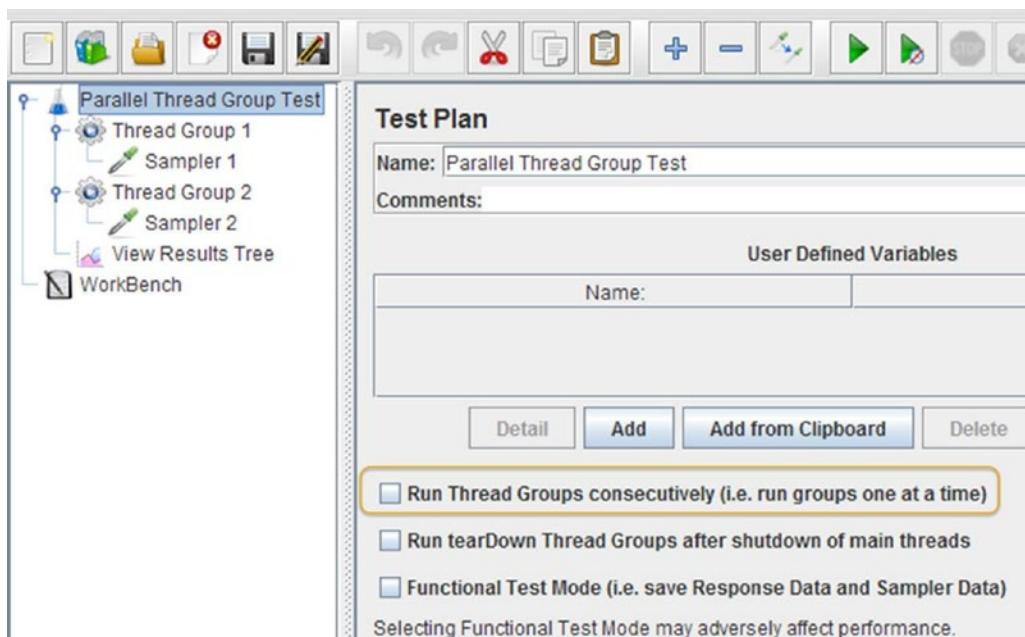


Figure 7-13. Uncheck the *Run Thread Groups Consecutively* option

2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group. Configure the name as **Thread Group 1** and the **Number of Threads (Users)** as 5.
3. Click on Thread Group and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /dt, and **Name** as Sampler 1.
4. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group. Configure **Name** as Thread Group 2 and **Number of Threads (users)** as 5.
5. Click on Thread Group and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /dt, and **Name** as Sampler 2.

⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_07/ParallelThreadGroupTestPlan.jmx

6. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
7. Save the test plan.
8. Run the test.

The results are shown in Figure 7-14.

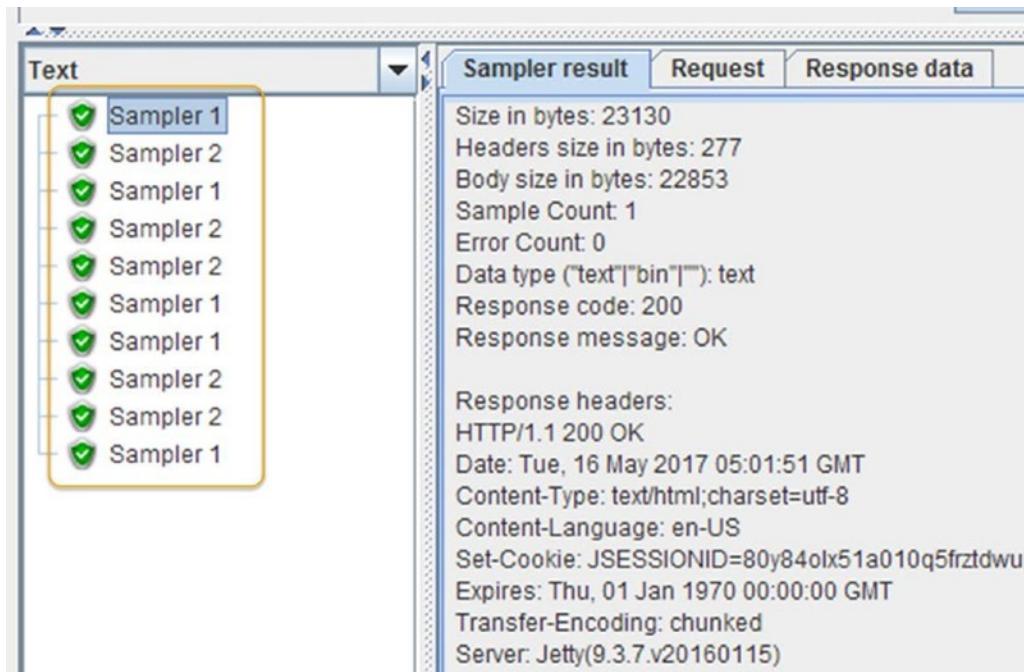


Figure 7-14. Parallel Threads sampler results

You can conclude that the thread groups have executed in parallel, as the web requests for Sampler 1 have been intermixed with requests for Sampler 2.

Using External File for Parameterizing User Login

In the real world, several different users are normally logged into a web application performing various operations. While doing performance testing, you'll want to mimic this behavior. This can be done using the parameterization of the login page and providing an external CSV file, which consists of a list of users arranged in columns, such as login-user and login-password. JMeter provides the *CSV Data Set Config* option to achieve this.

Follow these steps or download `CSVDataSetConfigTestPlan.jmx`.⁶

1. Create a test plan and give it a meaningful name, such as Using External CSV Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (Users)** as 5.

⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_07/CSVDataSetConfigTestPlan.jmx

3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Path** as /user/signIn, and **Method** as POST.
4. Click on HTTP Request and go to Edit ▶ Add ▶ Config Element. Add CSV Data Set Config. Configure **Filename** as users.csv, **Variable Names (comma-delimited)** as username,password, **Delimiter** as, (see Figure 7-15).

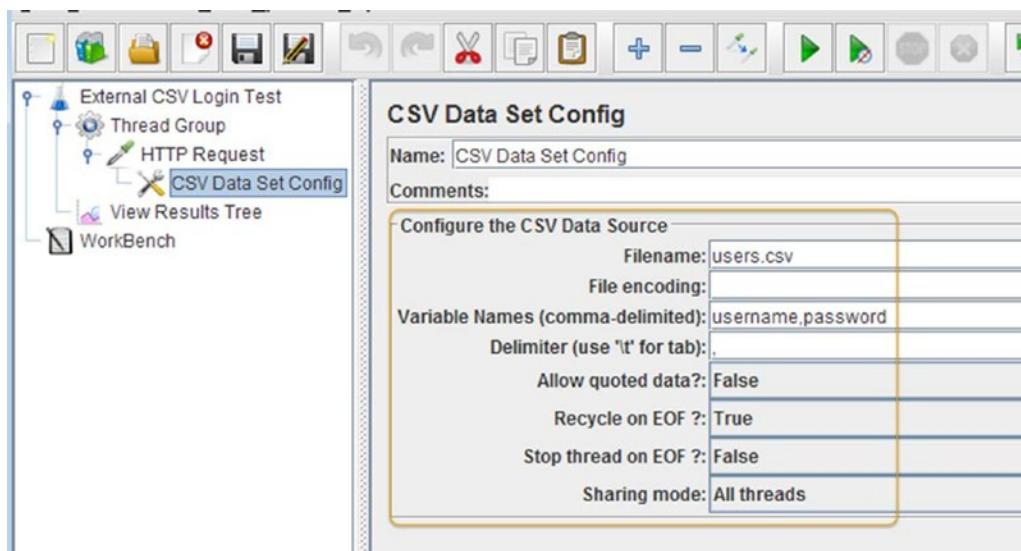


Figure 7-15. CSV Data Set Config

5. Click on HTTP Request and configure **Send Parameters with the Request**, as shown in Figure 7-16.



Figure 7-16. User login parameterization

6. Add the users.csv file to the same folder with these contents: user1@dt.com,user1 user2@dt.com,user2 user3@dt.com,user3 user4@dt.com,user4 user5@dt.com,user5.
7. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.

8. Save the test plan.
9. Run the test.

The results will be similar to those shown in Figure 7-17.

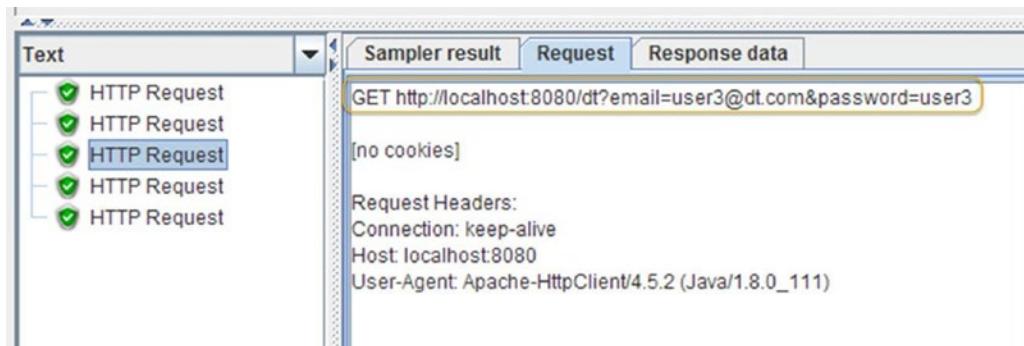


Figure 7-17. CSV data set config sampler results

Customizing Properties

JMeter can be customized by configuring various properties. By default, these are loaded from the `jmeter.properties` file. These can be overridden by a custom properties file.

It is a best practice to use your own custom properties file instead of modifying the `jmeter.properties` file. The custom properties file can be specified by modifying the `user.properties` property in the `jmeter.properties` file. By default, this property is set to `user.properties`.

The following snippet shows the custom properties being loaded from the `my.properties` file.

File: `jmeter.properties`

```
# Should JMeter automatically load additional JMeter properties? # File name to look for
# (comment to disable) user.properties=my.properties
```

Monitor JMeter Resource Usage

It is a best practice to monitor JMeter resource usage to ensure that tests are executed properly.

The resource usage information can be obtained from JMeter logs or by using a special tool like JVisualVM. This tool is shipped by default with JDK installation.

Refer to Chapter 8, “Troubleshooting JMeter,” to learn how to increase the HEAP size.

Standard Test Plan Templates

JMeter provides standard templates; see Figures 7-18 and 7-19. Using this, you can create quick standard test plans.

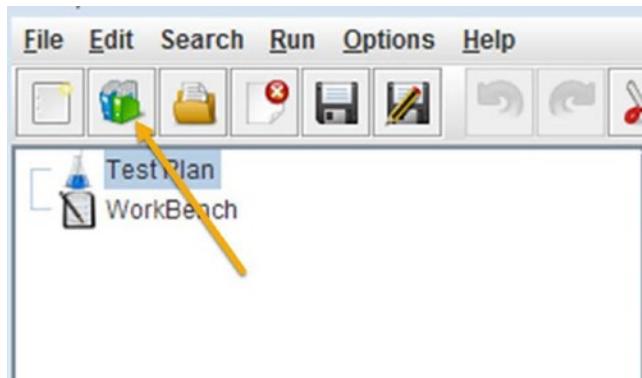


Figure 7-18. Standard test plan

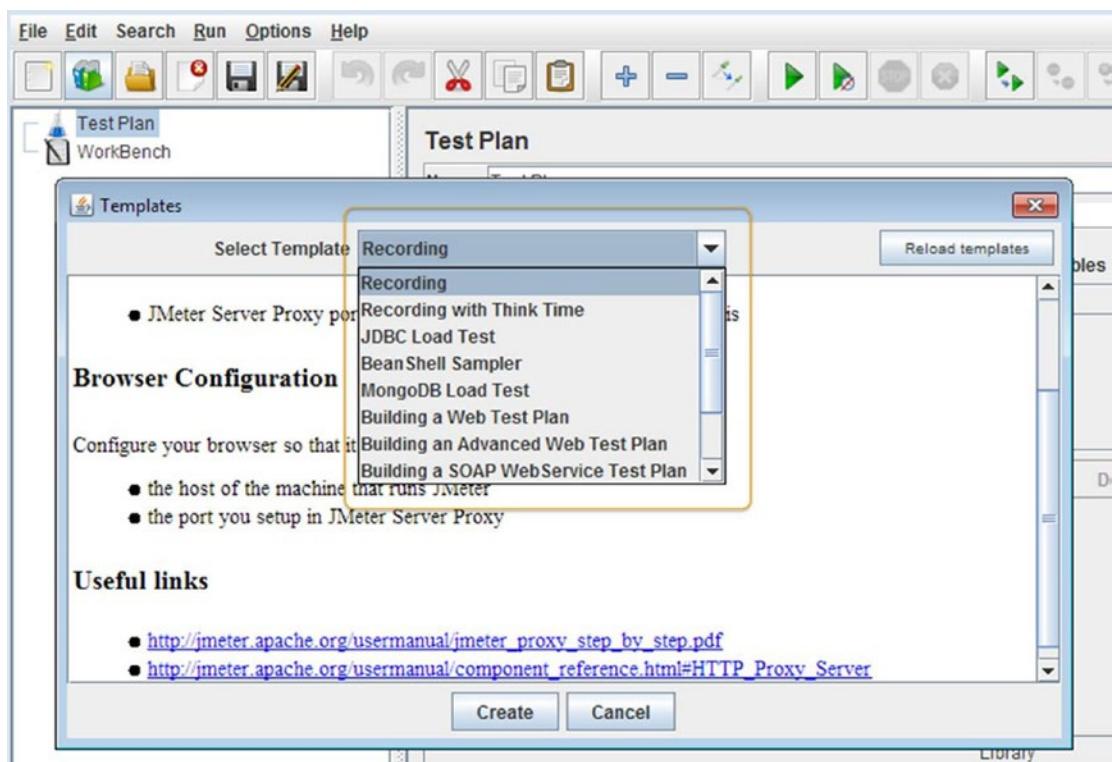


Figure 7-19. Standard test plan options

■ Tip It's good to have quick layout of the test plan, but make sure that you are not bound to standard templates and keep on innovating the test plan.

Conclusion

In this chapter, you learned few JMeter best practices that will help in keeping JMeter test scripts portable and ensure accurate results. In the next chapter, you learn how to troubleshoot various JMeter issues while developing JMeter test scripts.

CHAPTER 8



Troubleshooting JMeter

This chapter explains how to troubleshoot common JMeter issues while developing JMeter test scripts.

At the end of this chapter, you should have a good idea of troubleshooting various issues while developing JMeter scripts. It is a good idea to read this chapter unless you are an advanced user.

Ensure Permissions

For MacOS/Linux users, after installation, make sure that JMeter has the proper execute permissions. If you installed using an installer or expanded from a compressed tar (.tgz), then files will have the execute permission. However, if you have unzipped the binary distribution, the execute permission is not set.

On MacOSX/Linux, issue the following command.

```
cd <JMeter Installation Folder> cd bin chmod +x jmeter
```

Log File

The most obvious and important place to look for errors is in the log file. By default, the name of the log file is `jmeter.log` and is created in the directory in which you start JMeter. If you don't have write permissions on this directory, JMeter prints a `FileNotFoundException` error on the console. This issue occurs mostly with the MacOS and Linux systems. You will see the following error if that's the case.

```
$ ls -ld .
dr-x-----@ 12 saimatam  staff  408 Nov 18 22:32 .
```

```
$ jmeter
```

```
Writing log file to: /usr/local/apache-jmeter-3.0/ jmeter.log log_file=jmeter.log java.
io.FileNotFoundException: jmeter.log (Permission denied) [log_file-> System.out] 2016/06/23
10:29:06
INFO - jmeter.util.JMeterUtils: Setting Locale to en_US 2016/06/23 10:29:06
INFO - jmeter.JMeter: Loading user properties from: /usr/local/apache-jmeter-3.0/bin/user.properties
2016/06/23 10:29:06
INFO - jmeter.JMeter: Loading system properties from: /usr/local/apache-jmeter-3.0/bin/system.properties
2016/06/23 10:29:06
INFO - jmeter.JMeter: Copyright (c) 1998-2016 The Apache Software Foundation 2016/06/23
10:29:06
INFO - jmeter.JMeter: Version 3.0 r1743807
```

Sometimes, you'll want to specify a different name or a different directory for the log file. This is useful when you want to specify a custom directory for managing logs.

You can specify a different log file name or path by:

- Specify the `-j` option on the command line.
- Specify the `log_file` property in the `jmeter.properties` file.

In the following example, you specify the log file as `mylogfile`.

```
$ jmeter -j mylogfile
$ more mylogfile 2015/11/18 23:00:09 INFO  - jmeter.util.JMeterUtils: Setting Locale to
en_US
...
...
```

Log Level

By default, the JMeter logging level is set to `INFO`. However, you can change this to one of the following values: `FATAL_ERROR`, `ERROR`, `WARN`, `INFO`, and `DEBUG`.

The log level is predefined in the `jmeter.properties` file. Alternately, use the `L` option on the command line to specify the log level at the root level, as shown here.

```
jmeter -LDEBUG
```

Use the `-L` option on the command line to specify the log level at the package level, as shown here.

```
jmeter -Llog_level.jmeter.engine=DEBUG
```

In the `jmeter.properties` file, configure the log level by setting the properties. To specify the log level at the root level, use the following property:

```
File: jmeter.properties
log_level.jmeter=INFO
```

The level of logging for a package or individual class can be set by using the following format. Make sure that `org.apache` has been removed from the package name. The class name is optional.

```
log_level.[package_name].[classname]=[PRIORITY_LEVEL]
File: jmeter.properties
jmeter -Llog_level.jmeter.engine=DEBUG
```

The package names for the log levels are as follows:

File: `jmeter.properties`

```
log_level.jmeter=INFO
log_level.jmeter.junit=DEBUG
log_level.jmeter.control=DEBUG
log_level.jmeter.testbeans=DEBUG
log_level.jmeter.engine=DEBUG
log_level.jmeter.threads=DEBUG
log_level.jmeter.gui=WARN
```

```
log_level.jmeter.testelement=DEBUG
log_level.jmeter.util=WARN
log_level.jmeter.protocol.http=DEBUG
log_level.jmeter.protocol.http.control=DEBUG
log_level.jmeter.protocol.ftp=WARN
log_level.jmeter.protocol.jdbc=DEBUG
log_level.jmeter.protocol.java=WARN
log_level.jmeter.testelements.property=DEBUG
```

Note that `jorphan` is a root-level package and uses the following to set its log level.

```
log_level.jorphan=INFO
```

You can specify a different log file for different categories using `log_file`.

`[category]=[filename]`, where `category` is equivalent to the package/class name with `org.apache` omitted.

For example, to capture the logs from `jmeter.engine` into a file, you use the following property.

```
log_file.jmeter.engine=mylogs_engine.txt
```

HTTP Protocol Logs

`HttpClient` is an open source Java library used to start HTTP connections. This is the core of the `HTTP Request Sampler`. `HttpClient` was developed under the Apache Software Foundation. Apache and Commons `HttpClient` uses the same logging as JMeter. So if you enable debug logging on the `HttpClient`, logs will be captured under the JMeter log file.

To debug specific issues, you may want to configure the `HttpClient` to generate DEBUG logs.

```
log_level.jmeter.protocol.http=DEBUG
log_level.jmeter.protocol.http.control=DEBUG
```

You may also want to isolate these logs from the regular logs. To specify a separate file, you can use the following:

```
log_file.jmeter.protocol.http=mylogs_http.txt
```

GUI Logs

When JMeter starts in GUI mode, you'll find a Log Viewer panel at the bottom of the window. Initially, it may not be visible and can be enabled by a toggle. Click on the exclamation mark on a yellow triangle icon in the top-right corner of the toolbar.

Open `LogViewerPanelTest.jmx`.¹

Run the test and observe the log viewer panel (see Figure 8-1).

¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_08/LogViewerPanelTest.jmx

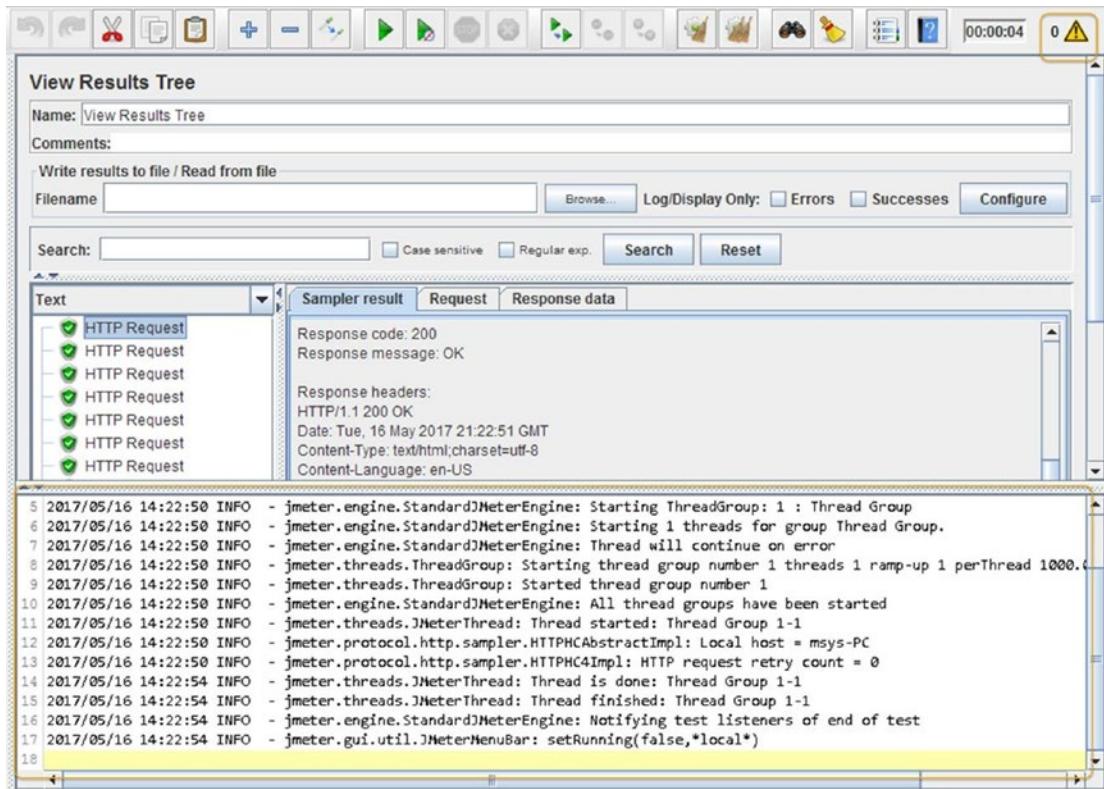


Figure 8-1. Log viewer panel

There is a `jmeter.loggerpanel.display` property in the `jmeter.properties` file. The default value is `false`. This determines if the Log Viewer Panel is initially displayed or hidden. Regardless of the value, you can click on the Toggle button in the Toolbar to enable/disable it.

We usually run JMeter in GUI mode during development and debugging of the test plan. GUI mode is not used when generating performance metrics for real use.

Clear GUI Logs

As you have seen in the previous section, after running the tests, you can see the logs by enabling the Log Viewer. If you have added listeners like *Aggregate*, *View Results Tree*, and other graphs, then after running the tests, these listeners will show the results. It would be good to clear these logs; navigate to Run ► Clear or Clear All to clear the logs.

Remote Host Exception

While using Linux VMs and starting remote hosts, you may get an error saying `java.RMI.RemoteException: Cannot Start. <vm hostname> is loopback address`. This is because under the `jmeter.properties` file, `remote_hosts` is set to `127.0.0.1`.

```
...
remote_hosts=127.0.0.1
#remote_hosts=localhost:1099,localhost:2010
...
```

And in the /etc/hosts file, the same 127.0.0.1 is set to localhost.

```
127.0.0.1 localhost
127.0.1.1 jagdeep-vm01
...
```

After removing the 127.0.0.1 localhost line from /etc/hosts, you will be able to start the remote jmeter-server.

There is another solution. Before starting jmeter-server, execute the following command.

```
$ export RMI_HOST_DEF=-Djava.rmi.server.hostname=<vm hostname>
```

After doing either of these two things, we will be able to start the remote jmeter-server properly.

This will not be a problem if you are using Windows, as the file C:\Windows\System32\Drivers\etc\hosts is commented.

Connect Exception

If one or more of the remote hosts are not reachable by the client, you will see the following message java.net.ConnectException: Operation timed out. Perhaps they have not yet booted up or they are shut down, in which case, when we trigger the test, it will fail.

For this example, we used FirstTestPlan.jmx.²

```
C:\>jmeter -n -t FirstTestPlan.jmx -R 192.168.0.7:1234,192.168.0.8:1234
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using FirstTestPlan.jmx
Configuring remote engine: 192.168.0.7:1234
Connection refused to host: 192.168.0.7; nested exception is:
    java.net.ConnectException: Connection refused: connect
Failed to configure 192.168.0.7:1234
Configuring remote engine: 192.168.0.8:1234
Connection refused to host: 192.168.0.8; nested exception is:
    java.net.ConnectException: Connection timed out: connect
Failed to configure 192.168.0.8:1234
Stopping remote engines
Remote engines have been stopped
Error in NonGUIDriver java.lang.RuntimeException: Following remote engines could not be
configured:[192.168.0.7:1234,
192.168.0.8:1234]
```

²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_03/FirstTestPlan.jmx

In the first case, when remote hosts are still booting up, JMeter has a property to wait for some time and then trigger the test. You can set how many retries JMeter has to make and how much retry delay it has to wait before starting the test.

By default, `client.tries` is set to 1 and `client.retries_delay` is set to 5000 milliseconds. Uncomment these properties and rerun the test.

```
# When distributed test is starting, there may be several attempts to initialize
# remote engines. By default, only single try is made. Increase following property
# to make it retry for additional times
client.tries=1

# If there is initialization retries, following property sets delay between attempts
client.retries_delay=5000
```

JMeter has a property to skip the remote hosts if they are not reachable and continue the test with the rest. Update `client.continue_on_fail` to true under the `jmeter.properties` file.

```
# When all initialization tries was made, test will fail if some remote engines are failed
# Set following property to true to ignore failed nodes and proceed with test
client.continue_on_fail=true
```

Run the test again. It will skip the unreachable host.

```
C:\>jmeter -n -t FirstTestPlan.jmx -R 192.168.0.7,192.168.0.8
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using FirstTestPlan.jmx
Configuring remote engine: 192.168.0.7
Configuring remote engine: 192.168.0.8
Connection refused to host: 192.168.0.8; nested exception is:
    java.net.ConnectException: Connection timed out: connect
Failed to configure 192.168.0.8
Following remote engines could not be configured:[192.168.0.8]
Continuing without failed engines...
Starting remote engines
Starting the test @ Mon May 15 17:04:14 PDT 2017 (1494893054551)
Remote engines have been started
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 2 in 00:00:01 = 2.2/s Avg: 376 Min: 279 Max: 474 Err: 0 (0.00%)
Tidying up remote @ Mon May 15 17:04:16 PDT 2017 (1494893056800)
... end of run
```

Solving Proxy Servers Problems

Sometimes, corporate networks have web proxy servers. Web traffic originating from inside the corporate network must go through the web proxy servers. One purpose is to conserve bandwidth by allowing only certain URLs or media types. The other important reason is security; unauthorized users or guests are prevented from connecting to the web by the `username/password` mechanism (see Figure 8-2).

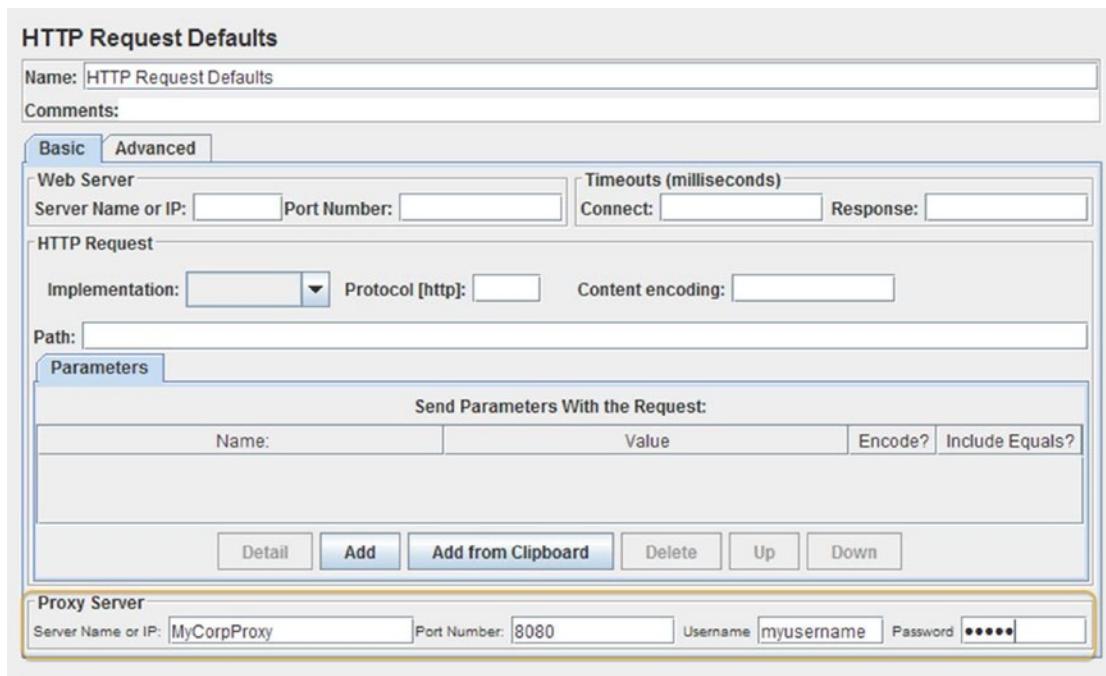


Figure 8-2. Proxy server configuration in HTTP request defaults

JMeter provides a simple way to specify proxy server details. This property is present in the *HTTP Request Defaults* component. If the JMeter test does not already have the *HTTP Request Defaults* component, add one under the Thread Group and apply it to the entire test plan.

Specify the Proxy Server details: Server Name or IP and Port Number.

Specify the username/password if the proxy server needs it. JMeter requests can now go through the proxy.

In Figure 8-2, the Proxy Server details are set as follows:

- **Server Name or IP** is set to MyCorpProxy
- **Username** is set to myusername
- **Password** is set to mypass

If you're using the command line and there is no username and password, leave them blank.

```
jmeter -H proxyserver -P 7000
```

If you're using the command line and the proxy server needs the username and password to authenticate, you use these additional parameters:

```
jmeter -H proxyserver -P 7000 -u username -a password
```

If you want to bypass the proxy and contact hosts directly, you can set the **-N** option as shown here:

```
jmeter -H proxyserver -P 7000 -u username -a password -N directHost
```

You can also use **--proxyHost**, **--proxyPort**, **--username**, and **--password** instead of **-H**, **-P**, **-u**, and **-a**.

HTTP Basic Authentication

As the title indicates, *HTTP Basic Authentication* is a very simple authentication mechanism used by the web server and the browser to secure access to URLs and resources. This does not need cookies, sessions, or login forms. The protocol is for the browser to send the username and password encoded in a variant of Base64 (RFC2045). This is not encrypted but merely encoded. It is straightforward to decode this. This is insecure, much like exchanging the username and password in plain text!

The protocol mechanism can be explained as follows:

1. The browser requests an URL or resource protected by the Basic Authentication mechanism.
2. The server responds with an HTTP status code of 401, which indicates an unauthorized request.
3. The server then responds with the `WWW-Authenticate` field in the header. For example:

```
WWW-Authenticate: Basic realm="abcd_m3VKxodZ2YM9:"
```

4. The client displays a dialog to the user to collect the necessary username and password, as shown in Figure 8-3.

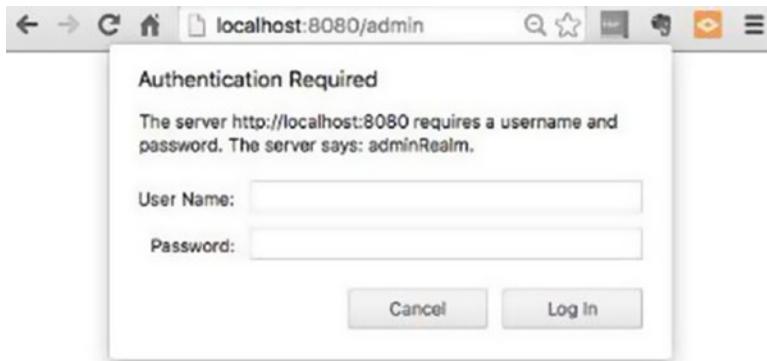


Figure 8-3. Authentication dialog displayed by the browser

5. The browser then encodes the string `username:password` using a variant of Base64-encoding (RFC2045) and sends it in an HTTP header field to the server.
6. The server authenticates and allows the request.
7. All the subsequent requests from the browser need to include the `username:password` encoding in the *HTTP Request*.

Using HTTP Header Manager

If the web application requires authentication, you can use the *HTTP Header Manager* to accomplish this.

Let's illustrate this with an example on how you should configure the *HTTP Header Manager* to run JMeter test scripts successfully.

Follow these steps or download `HeaderManagerTestPlan.jmx`.³

1. Create a test plan and give it a meaningful name, such as `Header Manager Test Plan`.
2. Click on `Test Plan` and go to `Edit > Add > Threads (Users)`. Add `Thread Group`.
3. Click on `Thread Group` and go to `Edit > Add > Config Element`. Add `HTTP Header Manager`.
4. Click on `Thread Group` and go to `Edit > Add > Sampler`. Add `HTTP Request`. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, and **Path** as `/admin`.
5. Click on `HTTP Request` and go to `Edit > Add > Assertions`. Add `Response Assertion`. Configure **Response Field to Test** as `Response Code`, **Pattern Matching Rules** as `Equals`, and **Patterns To Test** as `200`.
6. Click on `Thread Group` and go to `Edit > Add > Listener`. Add `View Results Tree`.
7. Save the test plan.

Now you need to configure the name and value for the `Headers Stored in Header Manager` option.

1. In the CMD prompt, issue the following command to encode the `user:password` string into the Base64 format.

```
echo -n "admin:admin" | base64 YWRtaW46YWRtaW4=
```

The string is `admin:admin` because the username and password are `admin`.

2. Add a header with the name of `Authorization` and a value of `BASIC YWRtaW46YWRtaW4=`, where `YWRtaW46YWRtaW4=` is the Base64-encoding obtained from the previous step (see Figure 8-4).

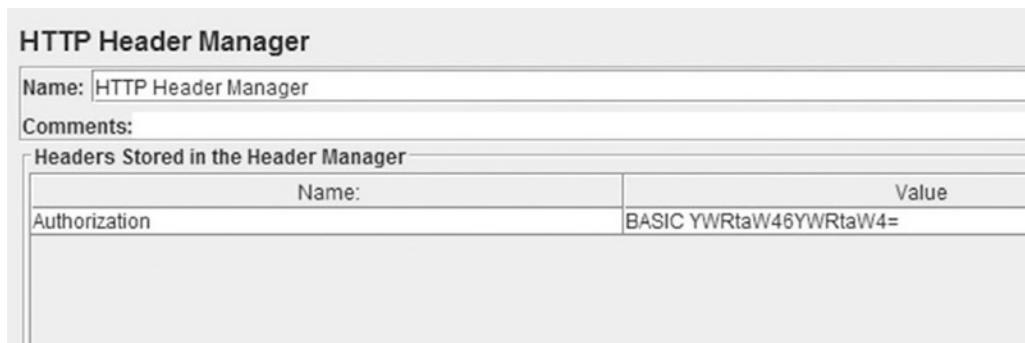


Figure 8-4. Header manager

³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_08/HeaderManagerTestPlan.jmx

- Run the JMeter test. It should be able to authenticate successfully.

Using the HTTP Authorization Manager

If the web application requires authentication, you can also use the *HTTP Authorization Manager* to accomplish this.

Let's illustrate this with an example of how you should configure the *HTTP Authorization Manager* to run JMeter test scripts successfully.

Follow these steps or download `AuthorizationManagerTestPlan.jmx`.⁴

- Create a test plan and give it a meaningful name, such as **Authorization Manager Test Plan**.
- Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
- Click on Thread Group and go to Edit ▶ Add ▶ Config Element and add HTTP Authorization Manager. Configure **Username** as `admin` and **Password** as `admin`. The **Base URL** may be configured based on the scope and the need. It is okay to omit the **Domain** and **Realm**. Leave the **Mechanism** as `BASIC_DIGEST` (see Figure 8-5).

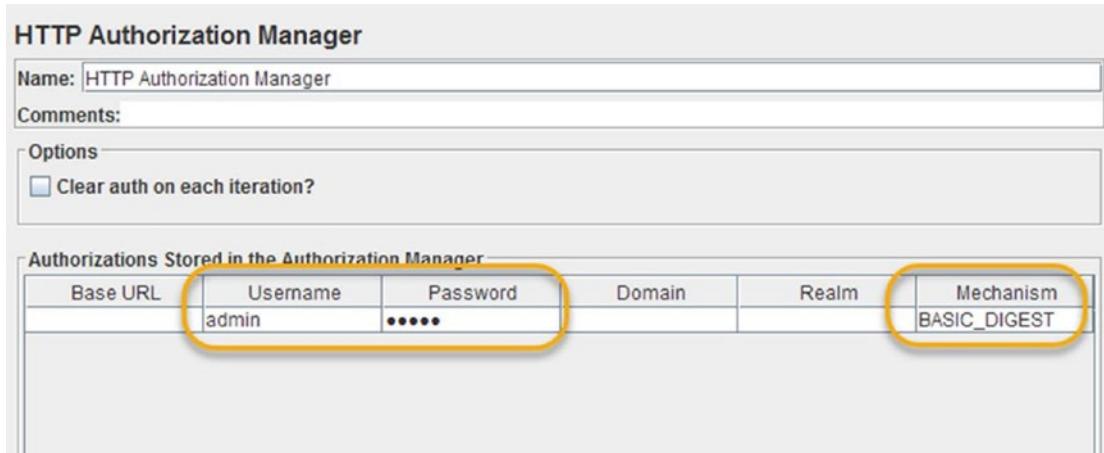


Figure 8-5. Authorization manager configuration

- Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as `localhost`, **Port Number** as `8080`, and **Path** as `/admin`.
- Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as `Response Code`, **Pattern Matching Rules** as `Equals`, and **Patterns To Test** as `200`.
- Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
- Save the test plan.

Run the JMeter test. It should be able to authenticate successfully.

⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_08/AuthorizationManagerTestPlan.jmx

Debug Test Faster

Prior to JMeter 3.0, we had to configure the thread group before you could run the test. With JMeter 3.0, you can run the test with a Validate menu option provided with the thread group.

The Validate menu option executes the specified thread group with a single thread and you can check the results quickly without having to configure thread group. This option is handy for debugging tests prior to the final run.

Let's illustrate this with an example of how you can use thread group Validate option.

Follow these steps or download `ThreadGroupValidateTestPlan.jmx`.⁵

1. Create a test plan and give it a meaningful name, such as thread group Validate Test Plan.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (Users)** to 10 and **Ramp-Up Period (in seconds)** to 10.
3. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /dt.
4. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Response Field to Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
6. Save the test plan.
7. Right-click on Thread Group and click on Validate (see Figure 8-6).

⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_08/ThreadGroupValidateTestPlan.jmx

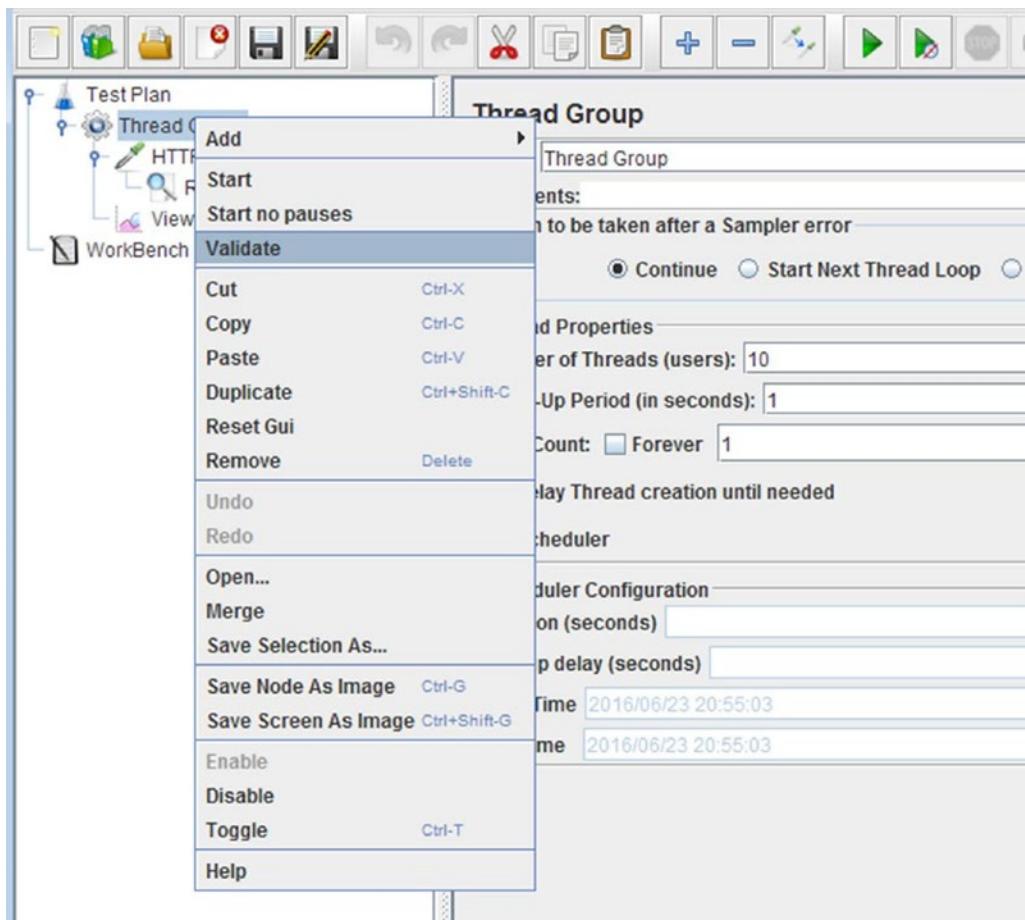


Figure 8-6. Thread group validate

8. Click on View Results Tree. You will observe that the test has been executed with a single thread even though it was configured with 10 threads.

There are two more options introduced with JMeter 3.0; Start and Start No Pause. These two options provide a shortcut to execute threads individually. The only difference between these two menu options is that with the latter, Timer if Configured in the thread group are skipped and the thread will run without any manual pauses.

These two options are useful when the test has multiple thread groups and you are still refining it. You can use these options to verify individual thread groups selectively.

Out of Memory Error

When JMeter exhausts its memory, you will notice the `java.lang.OutOfMemoryError` error in the logs.

```
C:\> jmeter -n -t DTPerformanceDashBoardTestPlan.jmx -l dt-pd.jtl -e -o C:\tmp\pd\
Writing log file to: jmeter.log
Creating summariser <summary>
Created the tree successfully using DTPerformanceDashBoardTestPlan.jmx
Starting the test @ Tue July 12 22:25:42 IST 2016 (1468342542968)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary + 192 in 00:00:17 = 11.4/s Avg: 11 Min: - Max: 187 Err: 0 (0.00%) Active: 46
Started: 46 Finished: 0
summary + 763 in 00:00:30 = 25.5/s Avg: 14 Min: - Max: 622 Err: 0 (0.00%) Active: 136
Started: 136 Finished: 0
summary = 956 in 00:00:47 = 20.4/s Avg: 13 Min: - Max: 622 Err: 0 (0.00%)
summary + 1474 in 00:00:30 = 49.2/s Avg: 20 Min: - Max: 428 Err: 0 (0.00%) Active: 226
Started: 226 Finished: 0
summary = 2430 in 00:00:17 = 31.6/s Avg: 17 Min: - Max: 622 Err: 0 (0.00%)
summary + 2000 in 00:00:30 = 65.6/s Avg: 174 Min: - Max: 1536 Err: 1 (0.05%) Active: 300
Started: 300 Finished: 0
summary = 4430 in 00:01:47 = 41.3/s Avg: 88 Min: - Max: 1536 Err: 0 (0.02%)
java.lang.OutOfMemoryError: GC overhead limit exceeded
Dumping heap to java_pid63753.hprof ...
Heap dump file created [127430909 bytes in 3.941 secs]
```

You can monitor the memory and CPU usage of JMeter using JVisualVM. This tool is shipped with the JDK installation, by default.

The JVisualVM screen shown in Figure 8-7 shows excessive CPU and memory usage by JMeter.

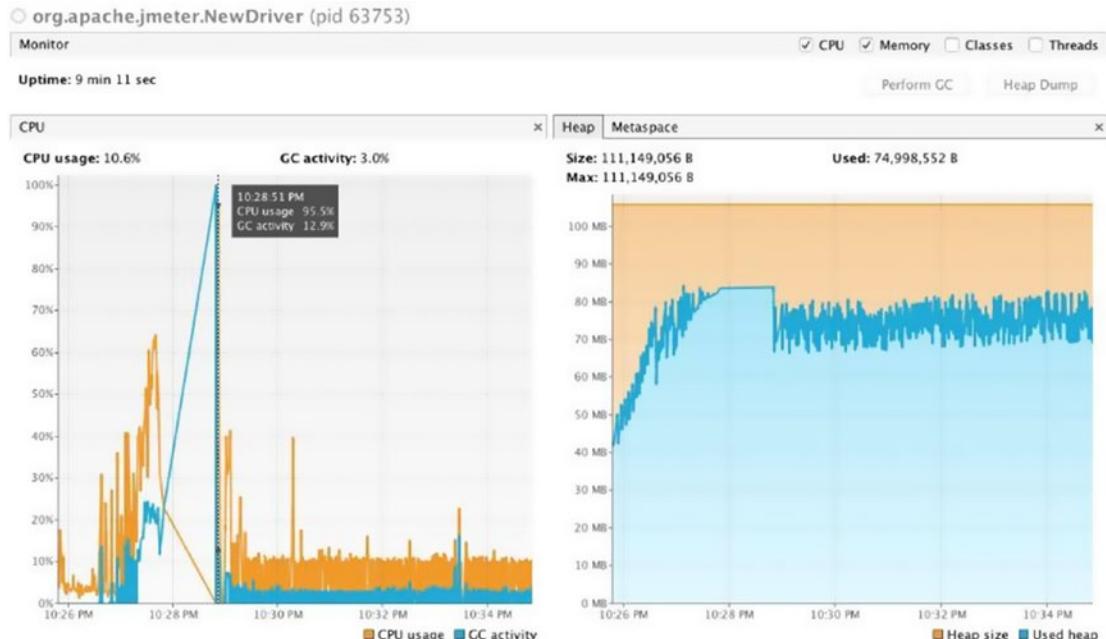


Figure 8-7. JVisual VM screen

To increase the HEAP memory for JMeter, specify it using the `JVM_ARGS` environment variable before starting JMeter. Enter this command in CMD prompt:

```
set JVM_ARGS="-Xms1024m -Xmx1024m"  
C:\> jmeter
```

Conclusion

In this chapter, you learned to troubleshoot various common errors while working with JMeter test scripts. In the next chapter, you will learn about JMeter plugins and learn how to use them in your test plan for monitoring and generating good looking reports.

CHAPTER 9



JMeter Plugins

This chapter introduces JMeter plugins, which are developed as a part of Google Code (JP@GC). They are useful in analyzing performance test results and displaying beautiful graphs. We will illustrate the use of PerfMon and jp@gc, the PerfMon Metrics Collector, with the help of an example.

At the end of this chapter, you will understand JMeter plugins, including how to install them and use them in test plans. You will also be able to develop and run test plans and export monitoring reports. Those who are already familiar with JMeter plugins can proceed to the next chapter.

Although JMeter generates very useful performance metrics, it does not have good visualizers and visually appealing graphs. *JMeter Plugins at Google Code (JP@GC)* closes this gap and offers several plugins that generate elegant graphs which help you visualize the performance test results. Additionally, JP@GC provides plugins that extend JMeter functionality with new config elements, timers, pre-processor, post-processor, assertions, listeners, and logic controllers.

For more details, look at JP@GC Standard Documentation.¹

Each JP@GC plugin has a *Help on This Plugin* link, which navigates to the Wiki page.² It also has an active community. You can also contact the people on the forum on the Mailing List.³

Each JP@GC plugin provides a rich set of options to customize the generation of pleasing graphs to meet a variety of needs.

PerfMon

During performance testing, it is important to know the health of the servers hosting the web application under test. To address this, the *PerfMon* package supports server/cluster monitoring. Using this, you can monitor CPU, memory, swap, disk I/O, and network performance on all platforms. See Figure 9-1.

¹<http://jmeter-plugins.org/wiki/Start/>

²<http://jmeter-plugins.org/wiki>

³<https://groups.google.com/forum/#!forum/jmeter-plugins>

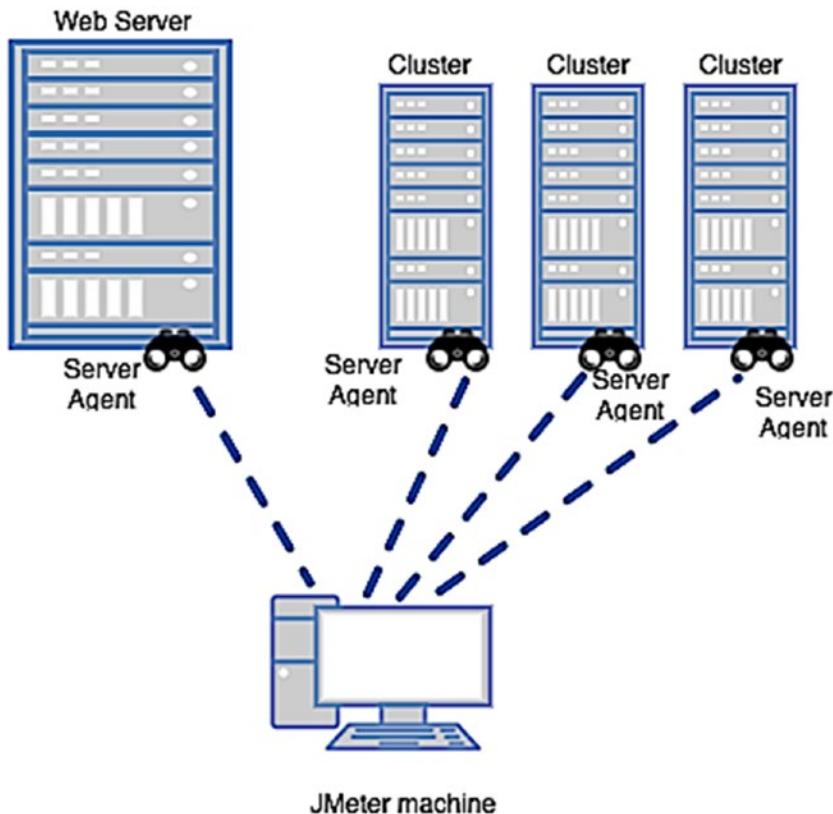


Figure 9-1. The server agent

To write this book, we used JMeter 3.0 and ServerAgent 2.2.1.

ServerAgent provides over 75 separate metrics, including CPU, memory metrics per-process, and custom metrics for measuring things such as file sizes, database row counts, Java heap sizes, and garbage collection time.

Download the Plugin

Download ServerAgent from the URL⁴ and set it up on the web application host.

Unzip ServerAgent-2.2.1.zip into a preferred folder. We downloaded it into the C:\ServerAgent-2.2.1 folder.

```
C:\ServerAgent-2.2.1>dir
Volume in drive C has no label.
Volume Serial Number is DA32-01EE
```

```
Directory of C:\ServerAgent-2.2.1
```

```
05/15/2017  10:36 PM    <DIR>      .
05/15/2017  10:36 PM    <DIR>      ..
```

⁴<http://jmeter-plugins.org/downloads/file/ServerAgent-2.2.1.zip>

```

02/25/2013 04:48 PM      10,821 CMDRunner.jar
05/15/2017 10:36 PM <DIR>          lib
02/25/2013 04:14 PM      85,433 LICENSE
02/25/2013 04:47 PM      62,848 ServerAgent.jar
02/25/2013 04:14 PM      63 startAgent.bat
02/25/2013 04:14 PM      74 startAgent.sh
      5 File(s)      159,239 bytes
      3 Dir(s)    7,551,549,440 bytes free

```

C:\ServerAgent-2.2.1>

Start the PerfMon Agent

To run the PerfMon agent, you will need JRE 1.4 or greater. PerfMon also gives you an option to run the agent in a specific JRE. Create a JRE folder under the root folder of the server agent and copy the JRE. Edit the `startAgent.bat` or `startAgent.sh` file and set the **JRE** path.

```

C:\ServerAgent-2.2.1>type startAgent.bat
@echo off
java -jar %0..\..\CMDRunner.jar --tool PerfMonAgent %*

```

Start the agent using the following command.

```

C:\ServerAgent-2.2.1>startAgent.bat
INFO    2017-05-15 22:40:13.101 [kg.apc.p] (): Binding UDP to 4444
INFO    2017-05-15 22:40:14.099 [kg.apc.p] (): Binding TCP to 4444
INFO    2017-05-15 22:40:14.101 [kg.apc.p] (): JP@GC Agent v2.2.0 started

```

By default, it opens UDP and TCP connections on port 4444, which are used to connect to JMeter to collect metrics. Console output indicates that the JP@GC server agent has been started successfully.

To start the server agent on a different port, specify the same on the command line.

The command-line options are `--udp-port` and `--tcp-port`.

```

C:\ServerAgent-2.2.1>startAgent.bat --udp-port 1234 --tcp-port 1234
INFO    2017-05-15 22:42:18.876 [kg.apc.p] (): Binding UDP to 1234
INFO    2017-05-15 22:42:19.874 [kg.apc.p] (): Binding TCP to 1234
INFO    2017-05-15 22:42:19.876 [kg.apc.p] (): JP@GC Agent v2.2.0 started

```

In the logs, you can see that UDP is bound to port 1234 and TCP is bound to 1235.

To stop the server agent, use this command:

C:\ServerAgent-2.2.1>`startAgent.bat --auto-shutdown`

Use the `--sysinfo` option to view available system objects.

```

C:\ServerAgent-2.2.1>startAgent.bat --sysinfo
INFO    2017-05-15 22:45:09.385 [kg.apc.p] (): *** Logging available processes ***
INFO    2017-05-15 22:45:09.475 [kg.apc.p] (): Process: pid=2080 name=taskhost.exe
      args=taskhost.exe
INFO    2017-05-15 22:45:09.477 [kg.apc.p] (): Process: pid=8980 name=dwm.exe
      args=C:\Windows\system32\Dll.dll
INFO    2017-05-15 22:45:09.478 [kg.apc.p] (): Process: pid=7572 name=explorer.exe
      args=C:\Windows\Explorer.EXE

```

```

INFO 2017-05-15 22:45:09.479 [kg.apc.p] (): Process: pid=6280 name=igfxem.exe
args=igfxEM.exe
INFO 2017-05-15 22:45:09.480 [kg.apc.p] (): Process: pid=6804 name=igfxhk.exe
args=igfxHK.exe
INFO 2017-05-15 22:45:09.481 [kg.apc.p] (): Process: pid=8640 name=igfxtray.exe
args=igfxTray.exe
...
...
INFO 2017-05-15 22:45:10.204 [kg.apc.p] (): Network interface: iface=eth35 addr=0.0.0.0
type=Ethernet
INFO 2017-05-15 22:45:10.207 [kg.apc.p] (): *** Done logging sysinfo ***
INFO 2017-05-15 22:45:10.210 [kg.apc.p] (): Binding UDP to 4444
INFO 2017-05-15 22:45:11.212 [kg.apc.p] (): Binding TCP to 4444
INFO 2017-05-15 22:45:11.217 [kg.apc.p] (): JP@GC Agent v2.2.0 started

```

You can use --loglevel parameter to set the level of logging. The log levels available are INFO, ERROR, WARNING, and DEBUG.

```

C:\ServerAgent-2.2.1>startAgent.bat --loglevel DEBUG
DEBUG 2017-05-15 22:47:19.760 [kg.apc.p] (): Start accepting connections
INFO 2017-05-15 22:47:19.823 [kg.apc.p] (): Binding UDP to 4444
INFO 2017-05-15 22:47:20.824 [kg.apc.p] (): Binding TCP to 4444
INFO 2017-05-15 22:47:20.825 [kg.apc.p] (): JP@GC Agent v2.2.0 startedConfigure PerfMon
Metrics Collector

```

You can download JMeter standard plugins from the JMeter Plugins web page.⁵

Extract the plugin into the apache-jmeter-3.0 root folder and execute the command as given. You will find the latest extracted files, as shown here.

```

c:\apache-jmeter-3.0\lib\ext>dir /O:-D
Volume in drive C has no label.
Volume Serial Number is DA32-01EE

Directory of c:\apache-jmeter-3.0\lib\ext

.
.
.

05/14/2016  11:47 AM           106 readme.txt
10/12/2015  10:07 AM      <DIR>          ..
10/12/2015  10:07 AM      <DIR>          .
10/12/2015  10:07 AM         10,782 CMDRunner.jar
10/12/2015  10:07 AM        1,262,178 JMeterPlugins-Standard.jar
06/17/2015  09:47 AM          95 JMeterPluginsCMD.sh
06/17/2015  09:47 AM          68 TestPlanCheck.bat
06/17/2015  09:47 AM          100 TestPlanCheck.sh
06/17/2015  09:47 AM          63 JMeterPluginsCMD.bat
               22 File(s)    4,387,697 bytes
               2 Dir(s)   7,541,547,008 bytes free

```

c:\apache-jmeter-3.0\lib\ext>

⁵<http://jmeter-plugins.org/downloads/file/JMeterPlugins-Standard-1.3.1.zip>

You can configure this plugin by using the following example.
Follow these steps or download PerfMonPluginTestPlan.jmx:⁶

1. Create a test plan and give it a meaningful name, such as PerfMon Plugin Test.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (users)** as 400 and **Loop Count** to 100.
3. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
4. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
5. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signIn and **Method** as POST.
6. Click on HTTP Request and under Parameters, add **Name/Value** as email/user1@dt.com. Add another row and configure password/user1.
7. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Path** as /user/signOut and **Method** as HEAD.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
9. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add jp@gc - PerfMon Metrics Collector (see Figure 9-2).

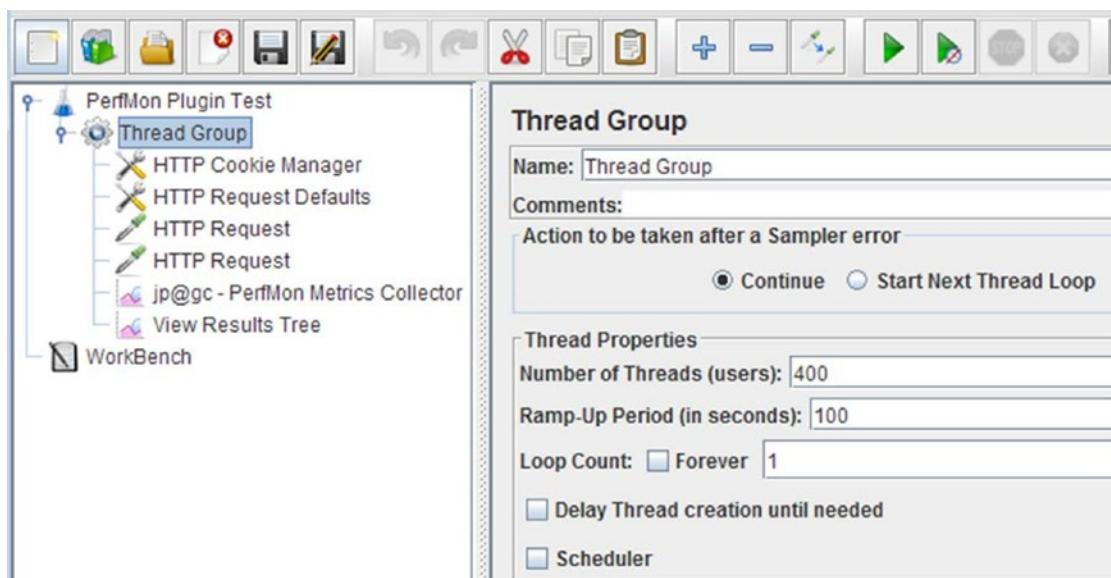


Figure 9-2. PerfMon plugin test

⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_09/PerfMonPluginTestPlan.jmx

10. Click on jp@gc - PerfMon Metrics Collector. Configure **Servers To Monitor** by clicking the Add Row button. Configure **HOST/ IP** as localhost, **Port** as 4444, and **Metric to Collect** as CPU (change it to meet your requirements).
11. Click again on the Add Row button to set **Metric to Collect** for Memory.
12. Save the test plan.
13. Start the server agent if it was not started earlier.
14. Run the test.

The results will be similar to those shown in Figure 9-3.

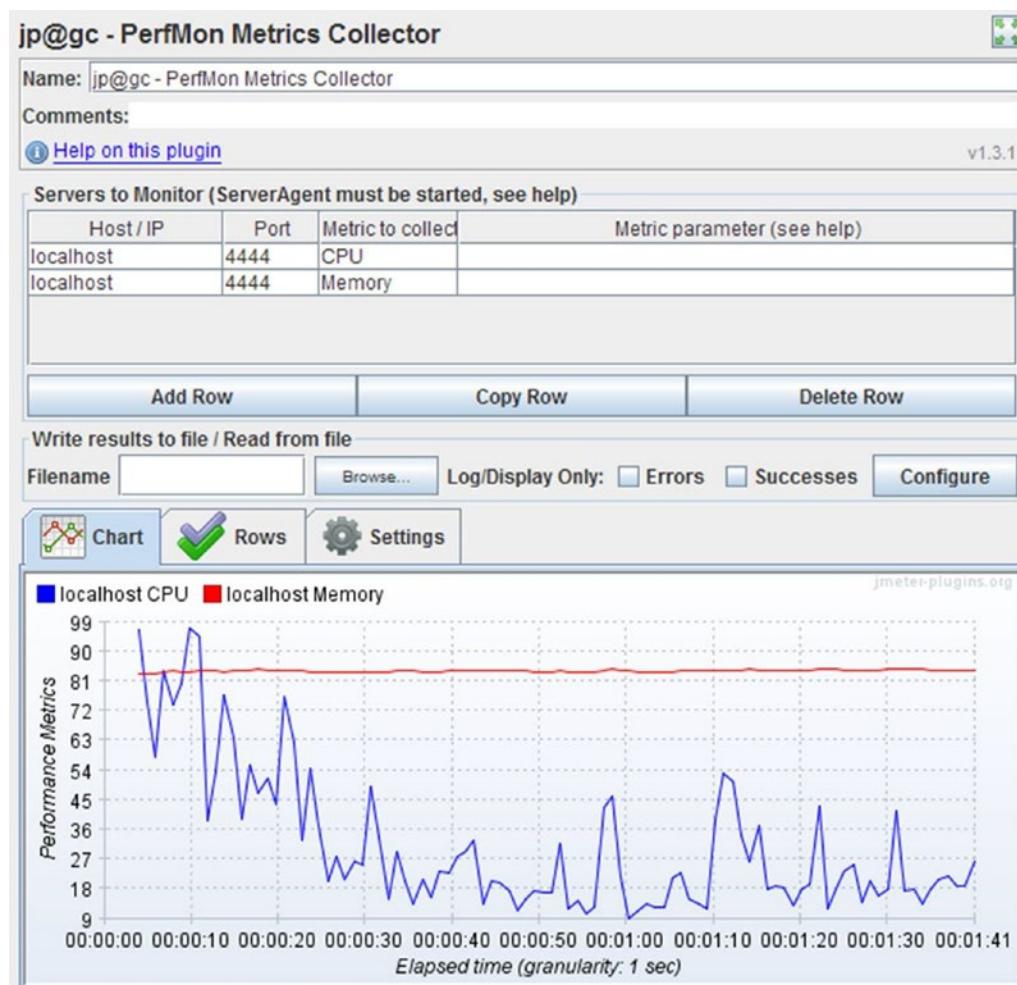


Figure 9-3. PerfMon metrics collector listener

To save the results as a graph, right-click on the graph and choose a suitable option (see Figure 9-4).

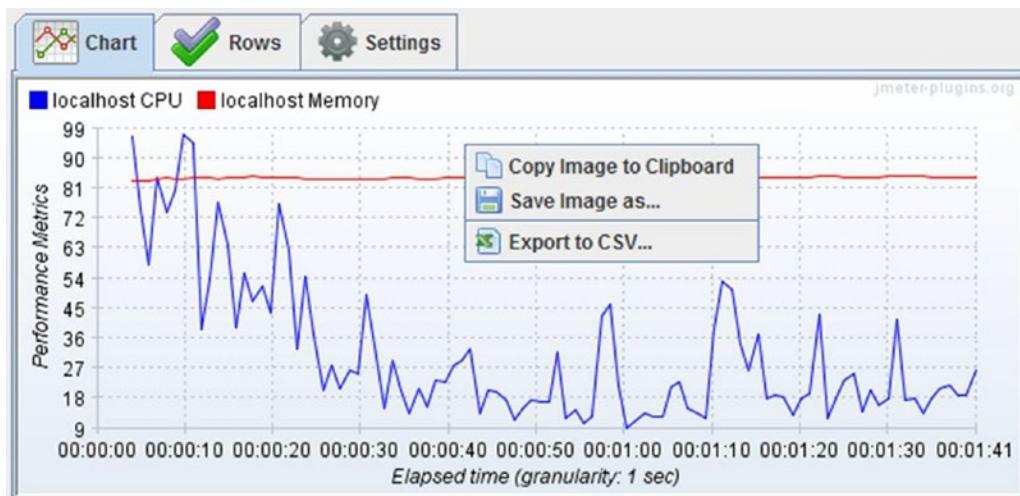


Figure 9-4. Save result graph

If you want to monitor different metrics on the same host/IP, click on **Copy Row** and change the Metric to Collect parameter. Available metrics are CPU, Memory, Swap, Disk I/O, Network I/O, TCP, JMX, EXEC, and TAIL. The Metric parameter can be used to set the CPU parameter per scope. Double-click on the cell, click on the three dots on the right side to open a new pop-up window. From that window, you can control the appearance of graphs. Mouse over to see the help text.

Figure 9-5 details the kind of metrics that can be collected for CPU.

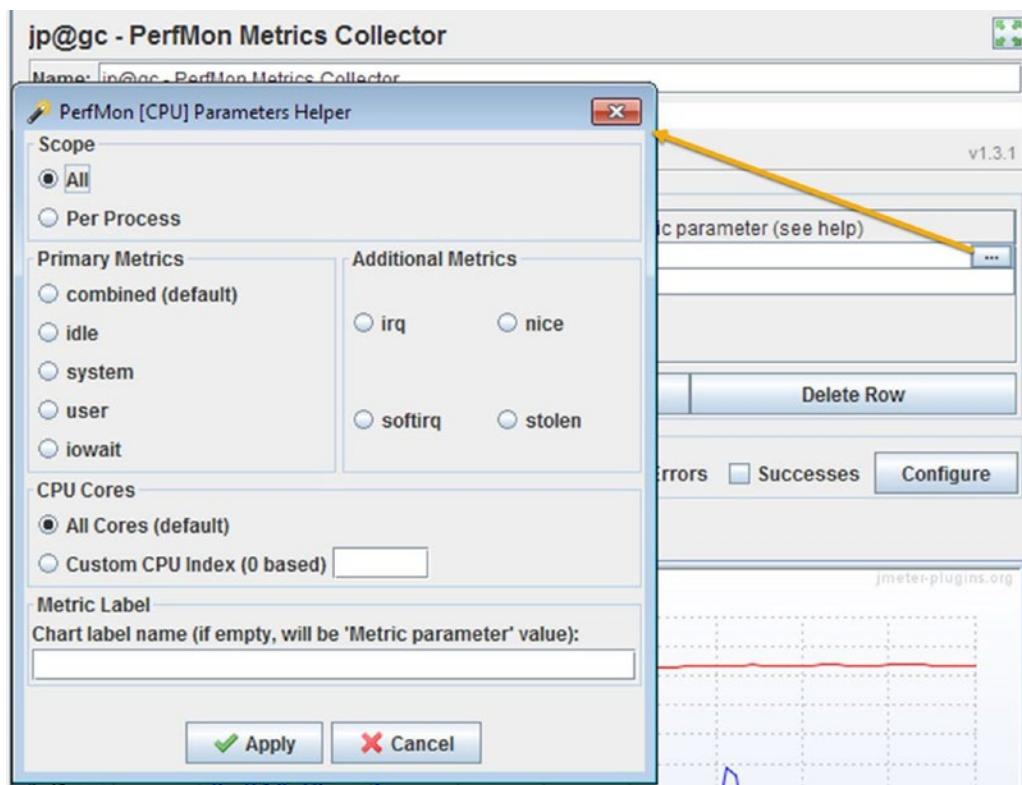


Figure 9-5. PerfMon CPU parameters helper

Non-GUI Mode

JMeter properties can also be used to configure the PerfMon plugin. To do this, add the following properties to the `jmeter.properties` file.

- `jmeterPlugin.perfmon.interval`: Metrics collection interval in milliseconds.
- `jmeterPlugin.perfmon.useUDP`: Set to true/false. Enabling this will make JMeter try a UDP connection if the TCP connection attempts fail.
- `jmeterPlugin.perfmon.label.useHostname`: Set to true/false. Enables using “short” hostnames. The default pattern is `([\w\-.]+)\..*`
- `jmeterPlugin.perfmon.label.useHostname.pattern`: String (escaped), regular expression to extract hostname (first group is matched).⁶ For example, the default pattern would be: `jmeterPlugin.perfmon.label.useHostname.pattern=([\w\-.]+)\..*`
 - Pattern for EC2 us-east/west subdomain matching is as follows: `jmeterPlugin.perfmon.label.useHostname.pattern=([\w\-.]+\..-(east|west)-[0-9]).*`
- `forcePerfmonFile`: Set to true/false. Enabling it forces JMeter to write a JTL file with PerfMon metrics in the current directory.

Run JMeter in non-GUI mode. Configure the filename to save the monitoring results too. Later, you can load the saved file into the GUI and review the timeline.

Conclusion

In this chapter, you learned about JMeter Plugins at Google Code (JP@GC) and explored various options for the *jp@gc - PerfMon Metrics Collector*. In the next chapter, you will learn about the advanced concepts of JMeter, which we call the JMeter recipes. These will be useful while performance testing a database/FTP/REST API/mobile web application.

CHAPTER 10



JMeter Recipes

This chapter explains advanced JMeter features like using JDBC, FTP, REST/JSON, AJAX, mobile web applications, and SOAP-XML performance testing, which will help you achieve specialized performance testing needs.

At the end of this chapter, you will have a good idea of a few advanced JMeter features and should be able to use these features to develop test scripts. Those who are already familiar with these features can proceed to the next chapter.

JDBC Performance Testing

This section covers a way to use JMeter to generate load and do the performance testing of a database running on MySQL.

Install MySQL

MySQL is a leading open source database server. Download and install the community server edition.¹

Follow the instructions on the web site to start the server.² Create a user and note the username and password. It is good to set the \bin folder in the standard path variable to access it from any directory.

```
C:\mysql-5.6.36-winx64\bin>mysqld
```

The previous command will start the mysql server. Use the following command to log in as the root.

```
C:\mysql-5.6.36-winx64\bin>mysql -u root  
mysql> create user "jmeter_user"@"localhost" identified by "mypass";  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> grant all on employees.* to "jmeter_user"@"localhost";  
Query OK, 0 rows affected (0.05 sec)  
  
mysql>
```

For this example, we will use Employees DB, which is a database that contains a large amount of data (approximately 160MB) spread over six separate tables and consisting of 4 million records in total. The documentation for this database is available on the MySQL web site.³

¹<https://dev.mysql.com/downloads/mysql/5.6.html#downloads>

²<https://dev.mysql.com/doc/refman/5.7/en/windows-start-command-line.html>

³<https://dev.mysql.com/doc/employee/en/>

Download a prepackaged archive of the data from LaunchPad.⁴

Unzip in the employee_db folder.

To set up Employees DB, open the command prompt and enter the following:

```
C:\employees_db>mysql -u jmeter_user -p -t < employees.sql  
Enter password: *****
```

Here is the output:

```
+-----+  
| INFO |  
+-----+  
| CREATING DATABASE STRUCTURE |  
+-----+  
+-----+  
| INFO |  
+-----+  
| storage engine: InnoDB |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING departments |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING employees |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING dept_emp |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING dept_manager |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING titles |  
+-----+  
+-----+  
| INFO |  
+-----+  
| LOADING salaries |  
+-----+
```

```
C:\employees_db>
```

⁴https://launchpad.net/test-db/employees-db-1/1.0.6/+download/employees_db-full-1.0.6.tar.bz2

Use the console to verify that the data loaded correctly.

```
C:\>mysql -u root;
Mysql> use employees;
mysql> select count(*) from employees;
+-----+
| count(*) |
+-----+
| 300024 |
+-----+
1 row in set (0.16 sec)
```

If you see this previous output, you have set up the database correctly.

The schema of the database is shown in Figure 10-1.

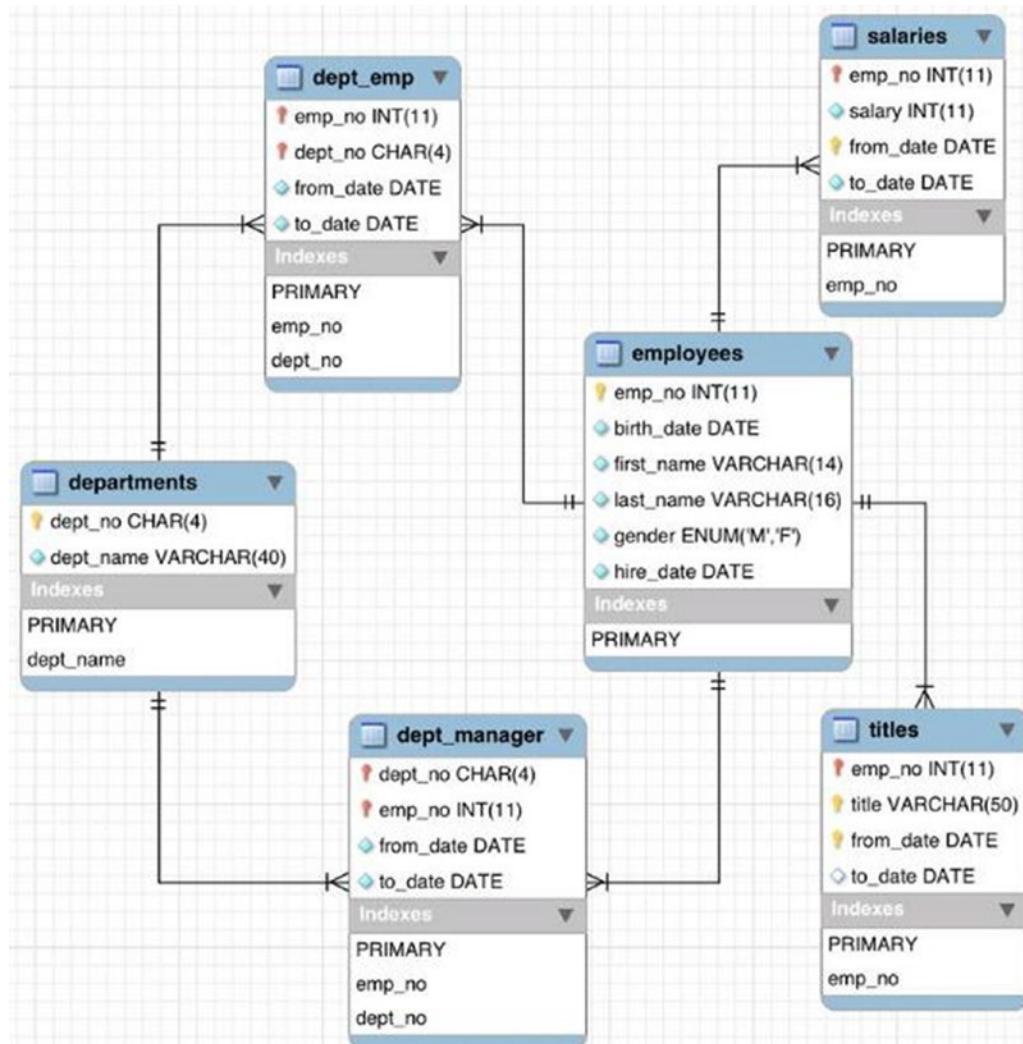


Figure 10-1. Employee schema

Install JDBC Driver

Download the MySQL JDBC connector.⁵

Unzip and copy the driver to the lib folder of JMeter.

JDBC Test Plan

The JDBC test plan is easy; you'll see by following an example.

Follow these steps or download `JDBCTestPlan.jmx`.⁶

1. Create a test plan and give it a meaningful name, such as `JDBC Test Plan`.
2. Click on Test Plan and go to `Edit > Add > Thread (Users)`. Add Thread Group. Configure **Number of Threads (Users)** as 1 and **Loop Count** as 1000.
3. Click on Thread Group and go to `Edit > Add > Configuration Element` to add JDBC Connection Configuration. Configure **Variable Name Bound to Pool**, **Variable Name** as `jdbcConfig`, **Database Connection Configuration**, **Database URL** as `jdbc:mysql://localhost:3306/employees`, **JDBC Driver class** as `com.mysql.jdbc.Driver`, **Username** as `jmeter_user`, and **Password** as created previously. Choose the defaults for the other parameters (see Figure 10-2).

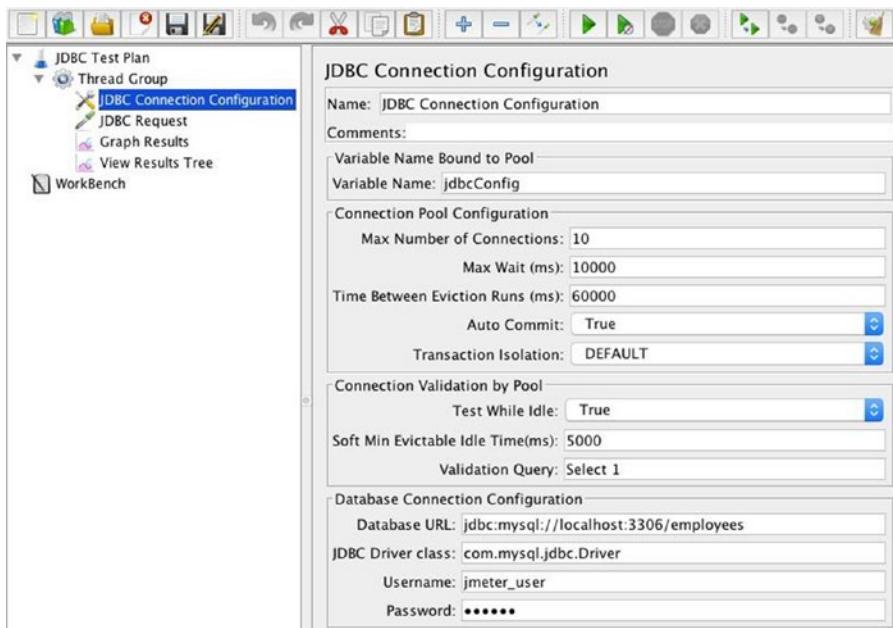


Figure 10-2. JDBC configuration

⁵<http://dev.mysql.com/downloads/connector/j/>

⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_10/JDBCTestPlan.jmx

- Click on Thread Group and go to Edit ▶ Add ▶ Samplers to add JDBC Request. Configure it with the **Variable Name Bound to the Pool** option set to `jdbcConfig` (this is the same value as configured in the JDBC Connection Configuration). Configure **Query** as `select count(*) from employee where first_name like 'Ge%'` (see Figure 10-3).

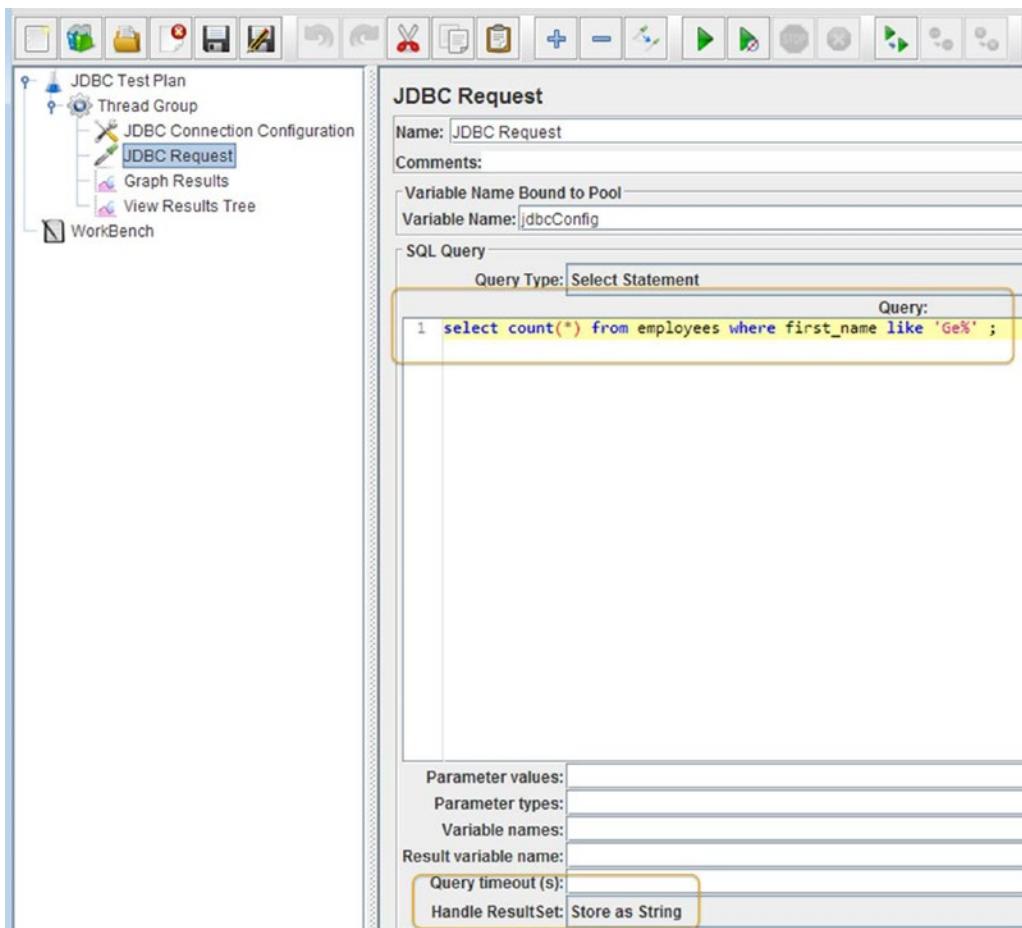


Figure 10-3. JDBC request sampler

- Run the test.

Results will be similar to those shown in the following figures.

Check the sampler results of the JDBC request (see Figure 10-4).

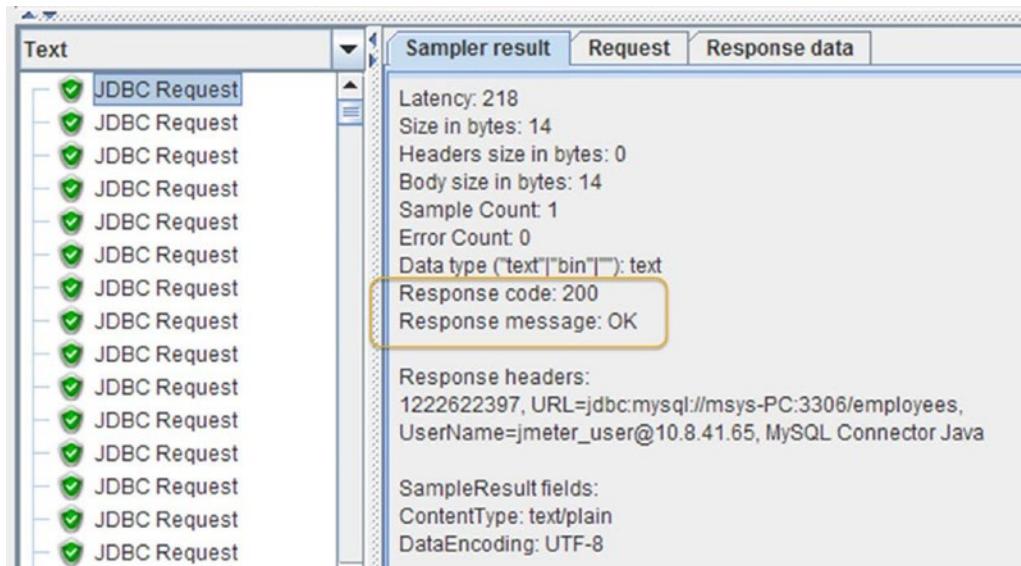


Figure 10-4. JDBC request results

Check the JDBC request that JMeter sent to MySQL (see Figure 10-5).

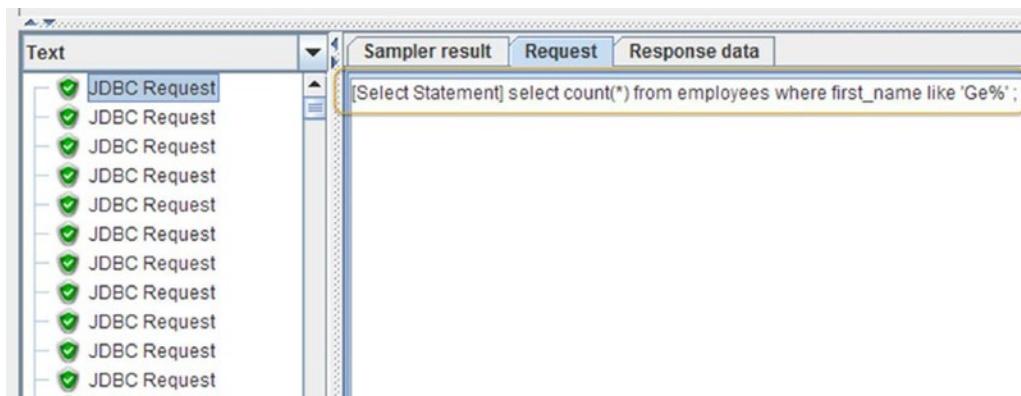


Figure 10-5. JDBC request SQL

Check the response data for the JDBC request (see Figure 10-6).

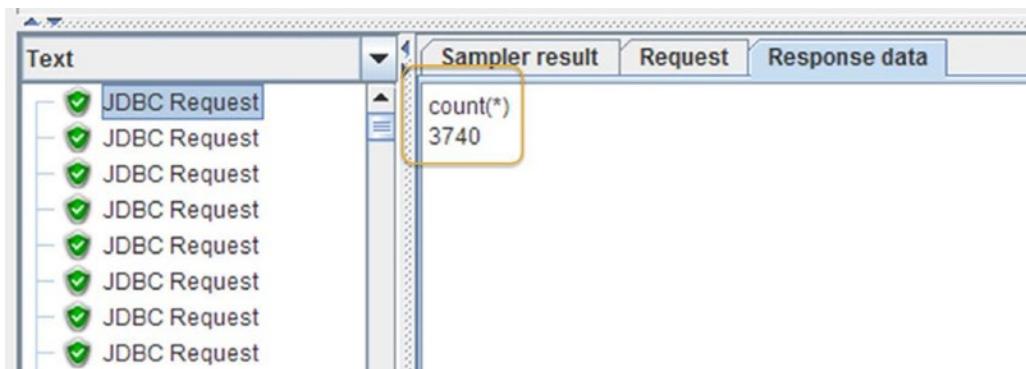


Figure 10-6. JDBC SQL Response

Look at the performance results as well.

Figure 10-7 shows the result of the test performed on the MySQL database.

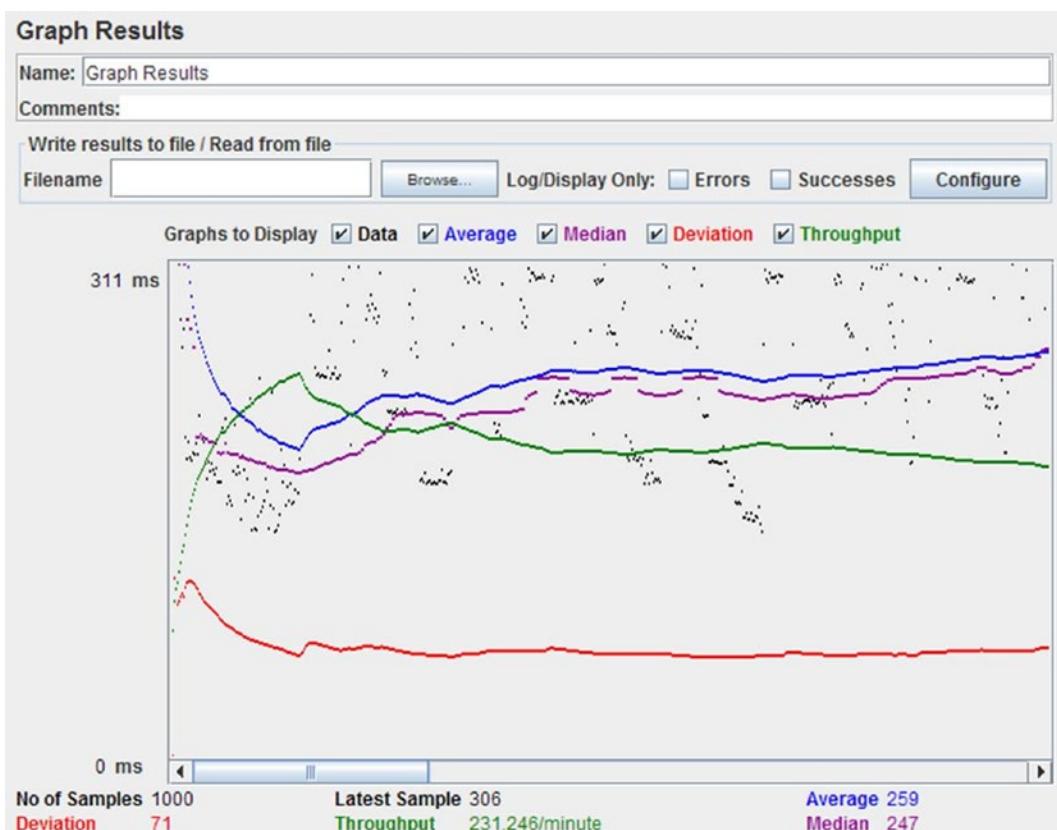


Figure 10-7. JDBC graph results

FTP Performance Testing

File Transfer Protocol (FTP) is a standard network protocol used to transfer files between computers. FTP follows a client-server model. The FTP client initiates a network connection (TCP) to the server. FTP uses separate control and data connections between the client and the server. FTP users authenticate themselves with a username and password, which is exchanged in clear text. Sometimes, the FTP server is configured to allow the client to connect anonymously without a password.

Tip FTP is commonly available, but it is considered insecure. Use SFTP instead.

Follow these steps to set up the FTP if it's not set up already.

1. Click on the Control Panel, then choose Programs and Features ➤ Turn Windows features on or off (left side). Select the Internet Information Service and select FTP Server, Web Management Tools, and World Wide Web Services. Click OK.
2. Click on the Control Panel, then choose ➤ Administrative Tools ➤ Internet Information Services (IIS) Manager. Right-click on the sites and then choose Add FTP Site.

Fill in the details. You are done setting up the FTP server locally.

3. Open your browser and launch `ftp://<your_ip_address>` enter user/password. You can see the local mapped folder files on the browser.

If you are Mac OS user, to start the FTP server, enter the following command in the terminal window.

```
$ sudo -s launchctl load -w /System/Library/LaunchDaemons/ ftp.plist
```

Note Be sure to shut down your FTP server after use for security reasons.

Let's look at an example to illustrate the performance testing of a FTP server. In this example, we will use JMeter to connect to the FTP server and transfer a file from the server. The file is an image named `cormorant.jpg`, so the transfer should use binary mode.

Follow these steps or download `FTPTestPlan.jmx`.⁷

1. Create a test plan and give it a meaningful name, such as `FTP Test Plan`.
2. Click on Test Plan and go to Edit ➤ Add ➤ Threads (users). Add Thread Group. Configure **Number of Threads (Users)** as `1` and **Loop Count** as `5000`.
3. Click on Thread Group and go to Edit ➤ Add ➤ Config Element ➤ FTP Request Defaults. Add FTP Request Defaults. Configure the **Server Name or IP** as `localhost` (see Figure 10-8).

⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_10/FTPTestPlan.jmx

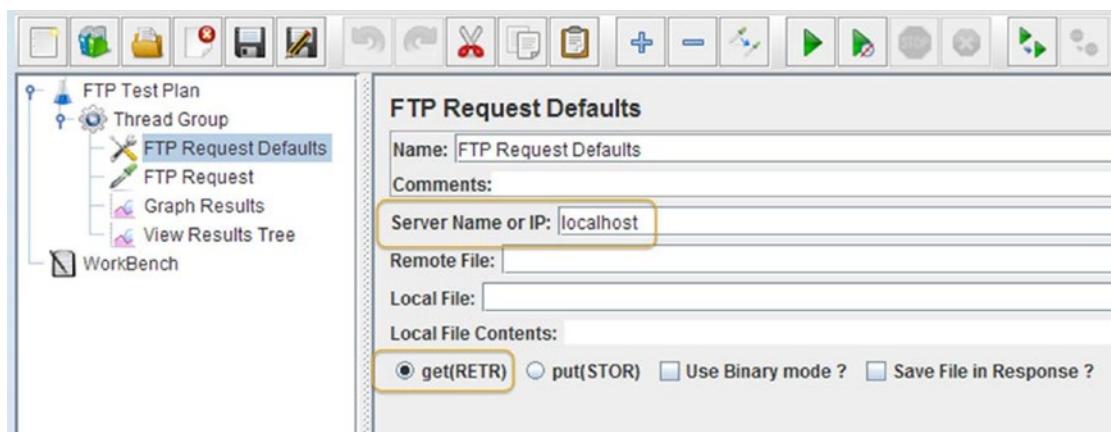


Figure 10-8. FTP request defaults

- Click on Thread Group and go to Edit ► Add ► Sampler. Add FTP Request. Configure the username and password. Set the value of the **Remote File** as /ftp.jpg and **Local File** as C:\temp\ftp.jpg. Since this is an image file, enable the Use Binary mode checkbox (see Figure 10-9).

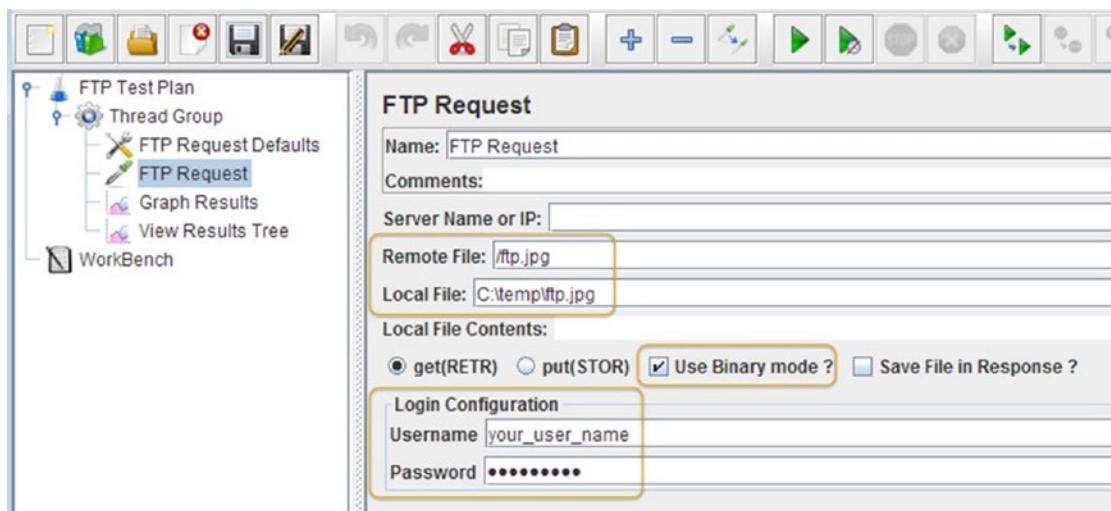


Figure 10-9. FTP request sampler

- Click on Thread Group and go to Edit ► Add ► Listener. Add Graph Result.
- Run the test.

Results will be similar to those shown in Figure 10-10.

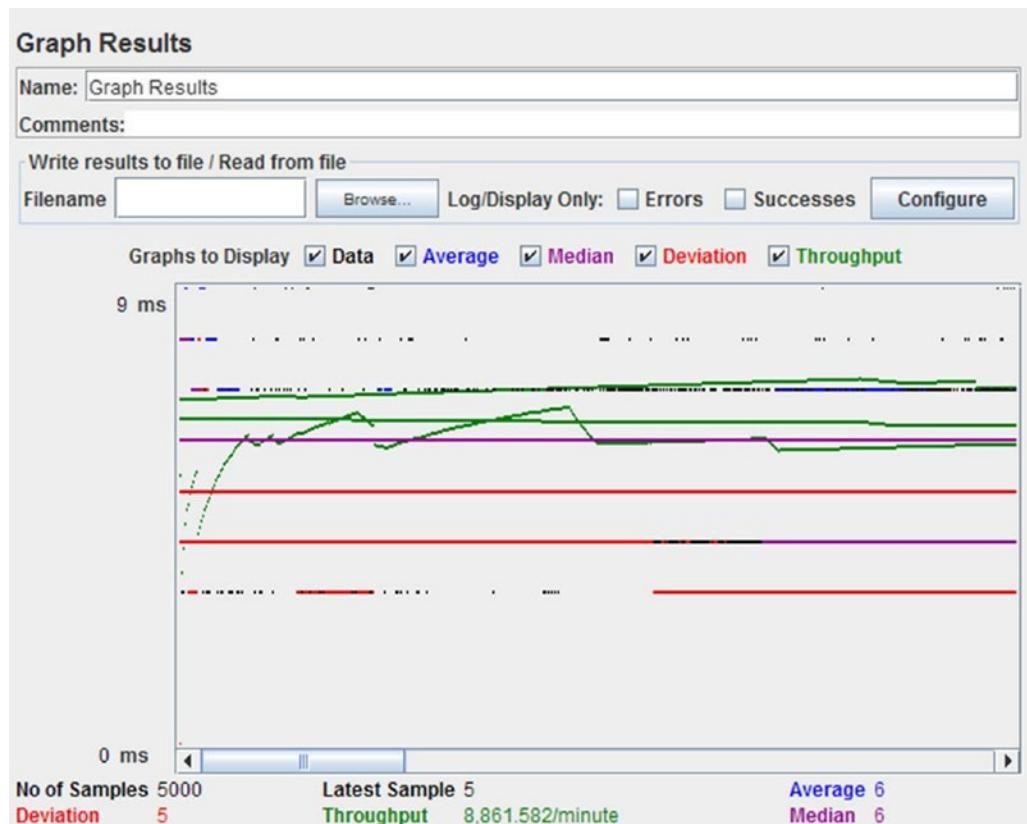


Figure 10-10. FTP request results

REST/JSON Performance Testing

REST stands for REpresentational State Transfer. It is not a specification or a protocol, but rather an architectural style. Through the use of REST architectural constraints, you can achieve performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

Roy Fielding introduced REST in 2000 in his doctoral dissertation at UC Irvine.⁸ He used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).

RESTful systems typically communicate over Hypertext Transfer Protocol (HTTP) with the HTTP verbs GET, POST, PUT, and DELETE. These verbs apply to web resources that are identified by Uniform Resource Identifiers (URIs).

Let's define the REST API for the entity called Book. Assume that it has the following attributes:

- title: Title of the book
- author: Book's primary author
- countryOfPublication: Country in which the book was published
- publishedBy: Book publishing company

⁸http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Now let's look at the REST API for Create, Read, Update, and Delete (CRUD) operations. The API uses the HTTP verbs:

- POST: Creates the book with a JSON payload
- GET: Reads the book given the ID
- PUT: Updates the book with a JSON payload
- DELETE: Deletes the book given the ID

Let's illustrate the Create operation through an example.

Follow these steps or download RESTJSONTestPlan.jmx.⁹

1. Create a test plan and give it a meaningful name, such as REST Create Book Test Plan.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
3. Click on Thread Group and go to Edit ▶ Add ▶ Configuration Element. Add HTTP Header Manager. Configure the Headers Stored in the Header Manager, **Name** as Content-Type and **Value** as application/json. Add another header with **Name** set to Accept and **Value** set to application/json (see Figure 10-11).

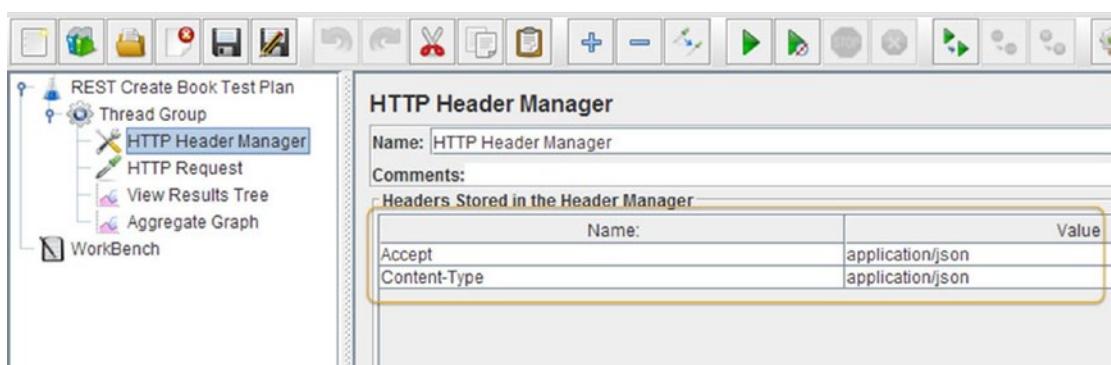


Figure 10-11. HTTP header manager

4. Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request Sampler. Configure **Server Name or IP** as localhost, **Port Number** as 8080, **Implementation** as HttpClient4, **Method** as POST, and **Path** as /books. Click on the body data and use the following JSON payload. It has required attributes for the Create operation. The ID has not been specified and it will be assigned when the book gets created (see Figure 10-12).

```
{
  "title": "The Zen Book",
  "countryOfPublication": "Japan",
  "publishedBy": "Good Books Inc.",
  "author": "Kobayashi"
}
```

⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_10/RESTJSONTestPlan.jmx

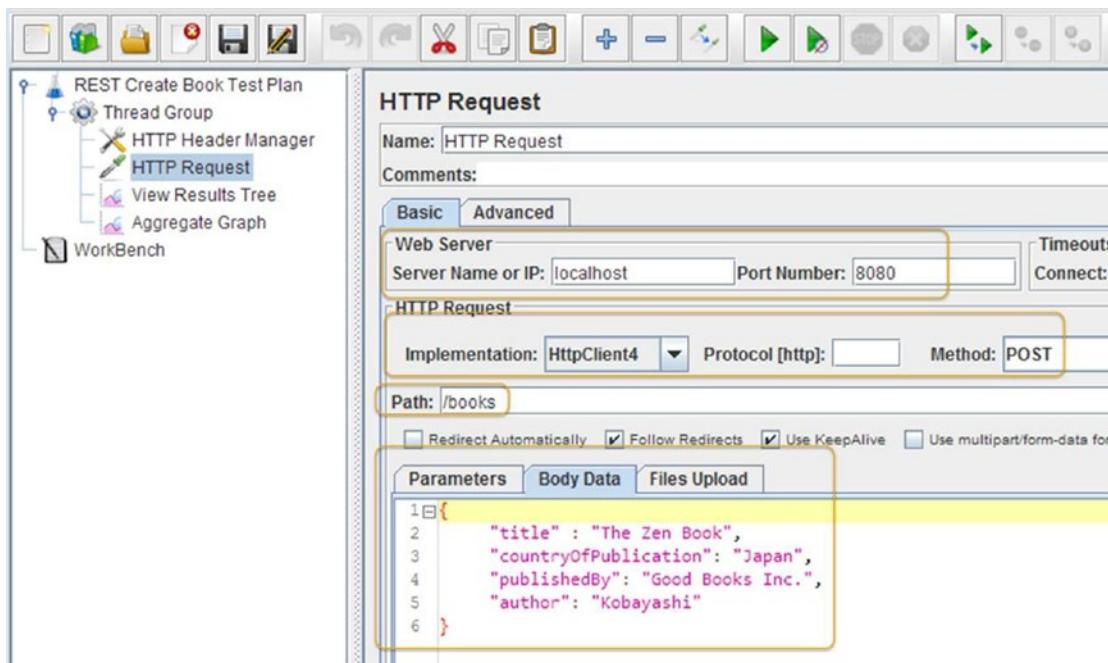


Figure 10-12. HTTP request

5. Click on Thread Group and go to Edit ▶ Add ▶ Listener to add View Results Tree.
6. Run the test.

Results will be similar to those shown in Figure 10-13.

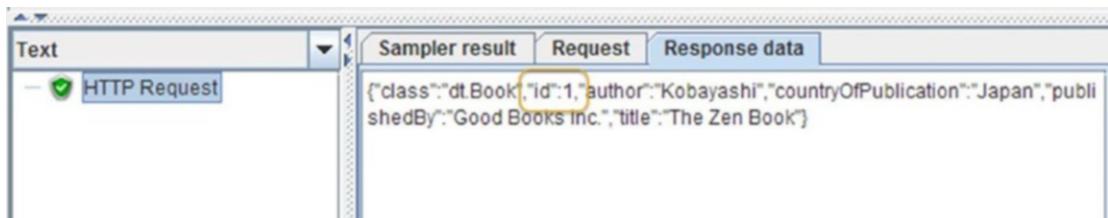


Figure 10-13. REST JSON response

Observe that the response has an ID attribute that was assigned by the server.

AJAX Performance Testing

AJAX stands for Asynchronous JavaScript and XML. AJAX's most important characteristic is its asynchronous nature of communication with the server. This lets you update portions of a page instead of having to refresh the entire web page. It can send/receive information in a variety of formats, including JSON, XML, HTML, and even text files.

Unlike the browser, JMeter does not execute JavaScript or render HTML. Testing AJAX is similar to any other web test. The key is to find all the AJAX requests and their expected responses. You can use the help of these tools/utilities/plugins:

- Firebug on Firefox¹⁰
- HttpWatch on Internet Explorer¹¹
- Developer tools in Chrome¹²

We use Chrome developer tools to illustrate. You can open the DevTools one of the following ways:

1. Select the Chrome menu at the top-right of your browser window, and then select Tools ➤ Developer Tools.
2. Right-click on any page element and select Inspect Element.
3. Press Ctrl+Shift+I (or Cmd+Opt+I on the Mac) to open DevTools.

The DevTools window will open at the bottom of your Chrome browser. Select the Network tab. Further, refine the selection by choosing only AJAX (XHR) requests. The AJAX request URLs are displayed on the left panel. When you click on a specific AJAX request URL, its request and response details are shown in the panel to the right.

Now you have the AJAX requests and responses to model your JMeter test, as shown in Figure 10-14.

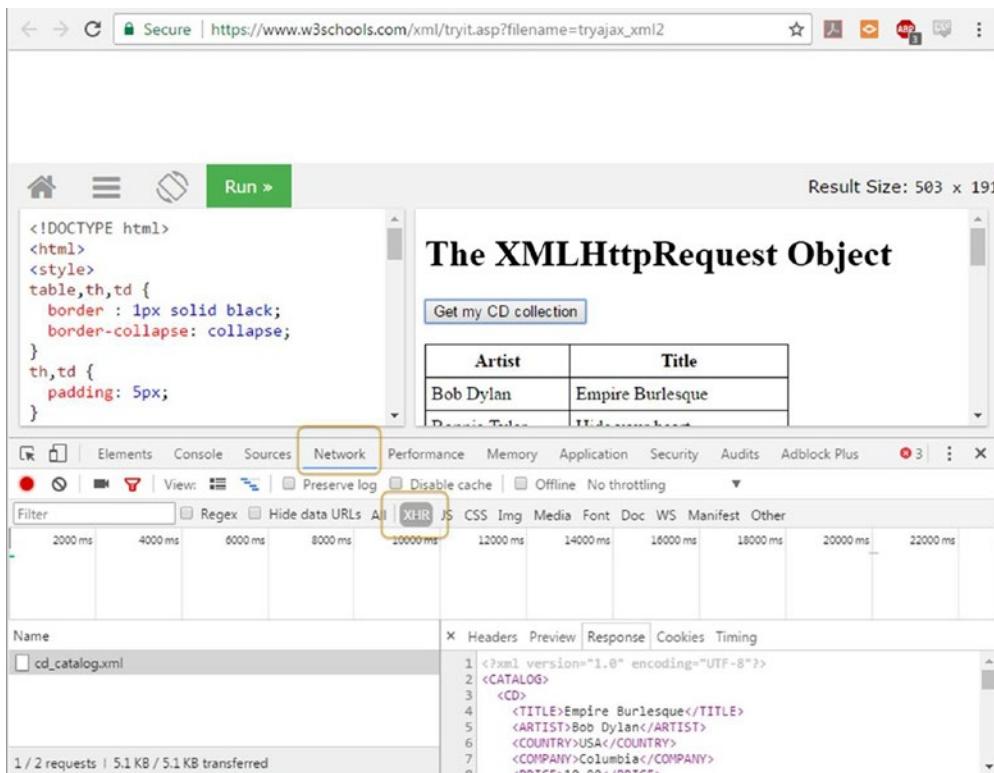


Figure 10-14. Chrome DevTools showing the AJAX requests

¹⁰<http://getfirebug.com>

¹¹<https://www.httpwatch.com>

¹²<https://developers.google.com/web/tools/chrome-devtools/>

Mobile Performance Testing

JMeter can be used to do the performance testing of web applications running on mobile devices.

Simulating Mobile Devices

When a browser, including the browser on a mobile device, interacts with the server, it includes a header called the User-Agent. So to pretend that the HTTP request originated on a mobile device/browser, you have to insert the same header value in your request.

You can find a list of User-Agent strings to use for every mobile device by searching the Internet.¹³ You can then use the JMeter HTTP Header Manager to send the appropriate User-Agent string.

The following example uses a User-Agent string of Mozilla/5.0 (iPad; CPU OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5355d Safari/8536.25, which corresponds to a Safari browser running on an iPad (see Figure 10-15).

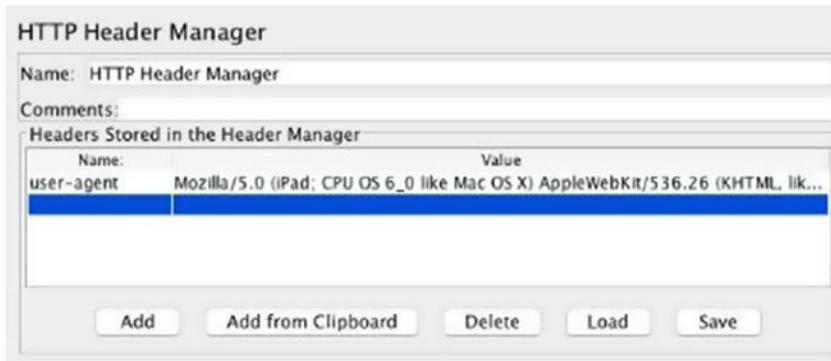


Figure 10-15. Header manager to set user agent

Simulating Network Speed

The connection speeds of mobile devices are often slow. It is easy to simulate slow connections.

The network speeds are measured using characters per second (cps), which can be calculated using this formula: $cps = (\text{target bandwidth in kbps} * 1024) / 8$

¹³<http://www.useragentstring.com/pages/useragentstring.php>

Select your device from the list in Table 10-1.¹⁴

Table 10-1. Device Bandwidth

Network	Bandwidth	CPS Value
Mobile data GPRS	171 kbit/s	21888
Mobile data EDGE	384 kbit/s	49152
Mobile data HSPA	14.4 Mbp/s	1843200
Mobile data HSPA+	21 Mbp/s	2688000
Mobile data DC-HSPA+	42 Mbps	5376000
Mobile data LTE	150 Mbp/s	19200000
WIFI 802.11a/g	54 Mbit/s	6912000
WIFI 802.11n	600 Mbit/s	76800000
Ethernet LAN	10 Mbit/s	1280000
Fast Ethernet	100 Mbit/s	12800000
Gigabit Ethernet	1 Gbit/s	128000000
10 Gigabit Ethernet	10 Gbit/s	1280000000
100 Gigabit Ethernet	100 Gbit/s	12800000000
WAN modems V.92 modems	56 kbit/s	7168
ADSL	8 Mbit/s	1024000
ADSL2	12 Mbit/s	1536000
ADSL2+	24 Mbit/s	3072000

The JMeter property file `jmeter.properties` has these two properties that you can configure.

```
httpclient.socket.http.cps=0
httpclient.socket.https.cps=0
```

Specify these values in the `JMETER_HOME/bin/user.properties` file.

JMeter to Record User Actions

Earlier in the book, we explored how to use JMeter to record a test script. You could use the same technique to record tests for mobile devices. The only difference is that you configure the mobile device to use JMeter as a proxy.

¹⁴Credit: Antonio Gomes Rodrigues

Android Proxy Configuration

1. Go to Settings > Wi-Fi.
2. Long tap on the connected network and click the Modify Network option.
3. From the dialog box, check the Advanced Options checkbox.
4. This will open advanced settings from which you can modify the proxy manually. For this, set the Proxy option to Manual.
5. Now set the **Proxy hostname** to your computer's IP address and set the **Proxy port** to 8080 (see Figure 10-16).

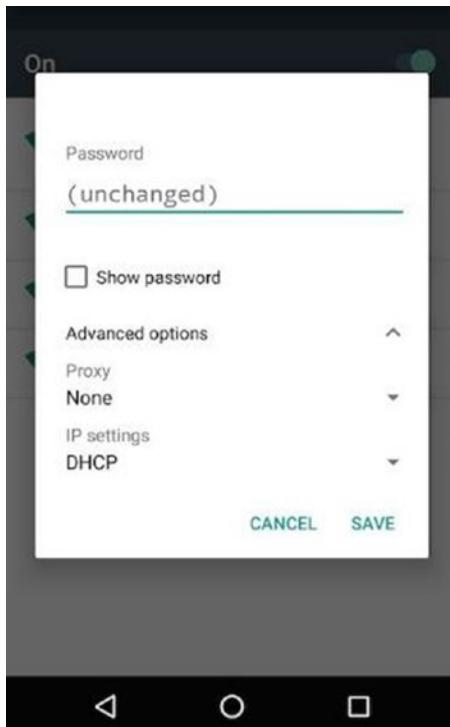


Figure 10-16. Android proxy settings

6. Click on the Save option. Run the application on your mobile device. Its requests will be recorded in JMeter.

iOS Proxy Configuration

1. Go to Settings > Wi-Fi.
2. Click on your connected network.
3. Select the Manual option from the HTTP Proxy section.

- Set the **Server** value to your computer's IP address and the **Port** value to 8080.
- Click on the Save option. Run the application on your mobile device. Its requests will be recorded in Jmeter, as shown in Figure 10-17.

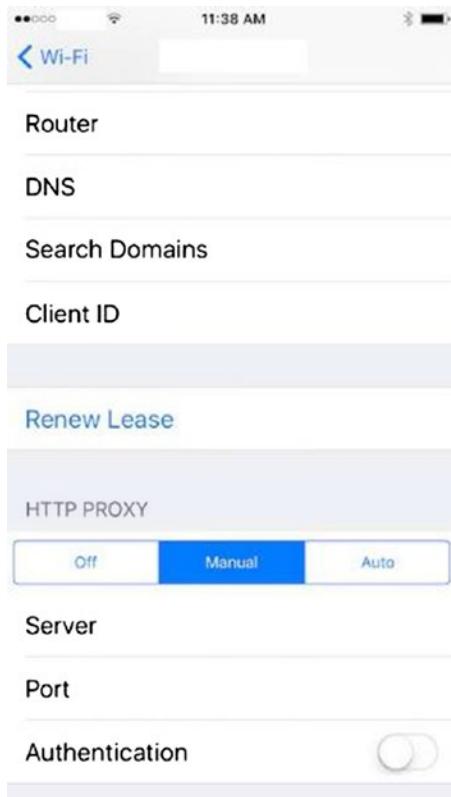


Figure 10-17. IOS proxy settings

SOAP Performance Testing

SOAP stands for Simple Object Access Protocol. This is a specification for exchanging structured information in the XML message format and relies on application layer protocols, most notably Hypertext Transfer Protocol (HTTP) for message negotiation and transmission. SMTP and JMS are other possible transport mechanisms.

SOAP was the desired mechanism to implement the Service Oriented Architecture (SOA). This has largely been replaced with REST.

W3C has a SOAP specification.¹⁵

¹⁵https://www.w3.org/TR/#tr_SOAP

Install SOAPUI

SoapUI is a free and open source functional testing tool with very good support for SOAP. Download the SoapUI package and then follow these steps to create a new project.¹⁶

1. Go to File ➤ New to create a new project. Choose the **Description File** and specify the value as Initial WSDL and URL as `http://localhost:8080/services/CurrencyService?WSDL`. Enable the Create Sample Requests for All Operations checkbox to automatically generate sample SOAP requests (see Figure 10-18).

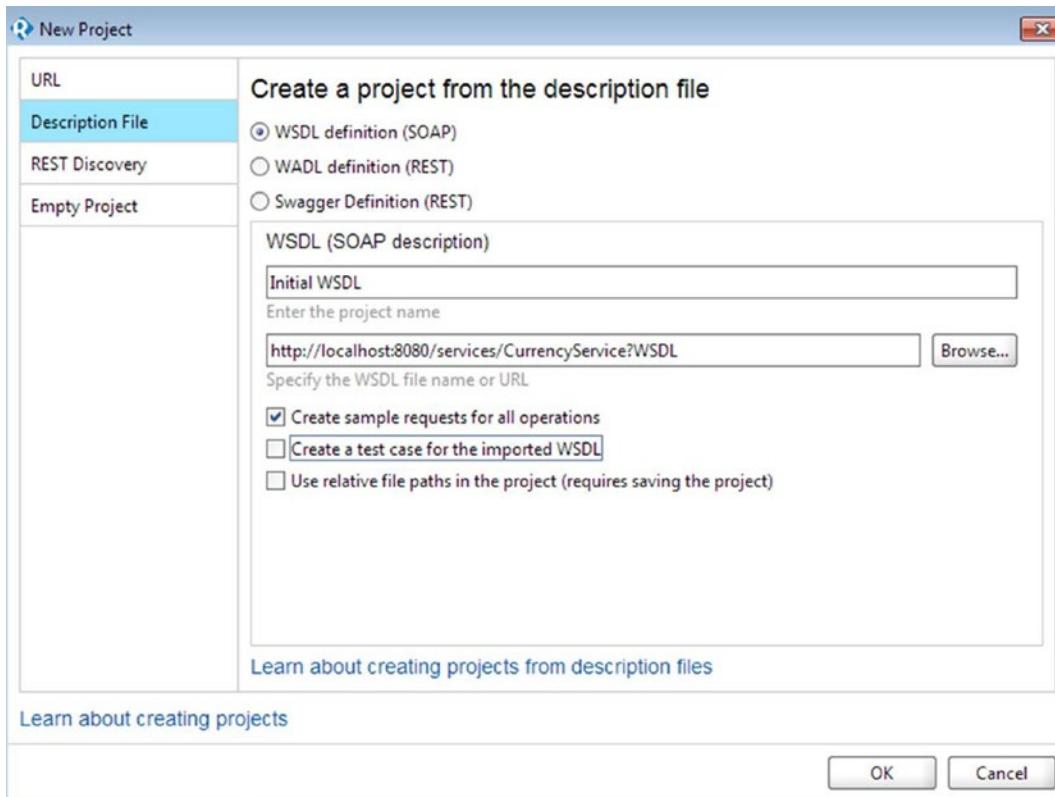


Figure 10-18. New SOAP project

¹⁶<http://www.soapui.org>

- Double-click on Request1 to open the request template on the pane to the right (see Figure 10-19).

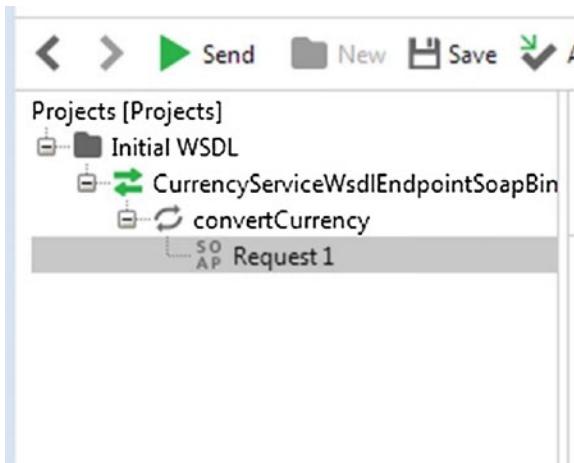


Figure 10-19. SOAP request menu

- Fill in the XML arguments. Set **arg0** to USD and **arg1** to JPY (see Figure 10-20).

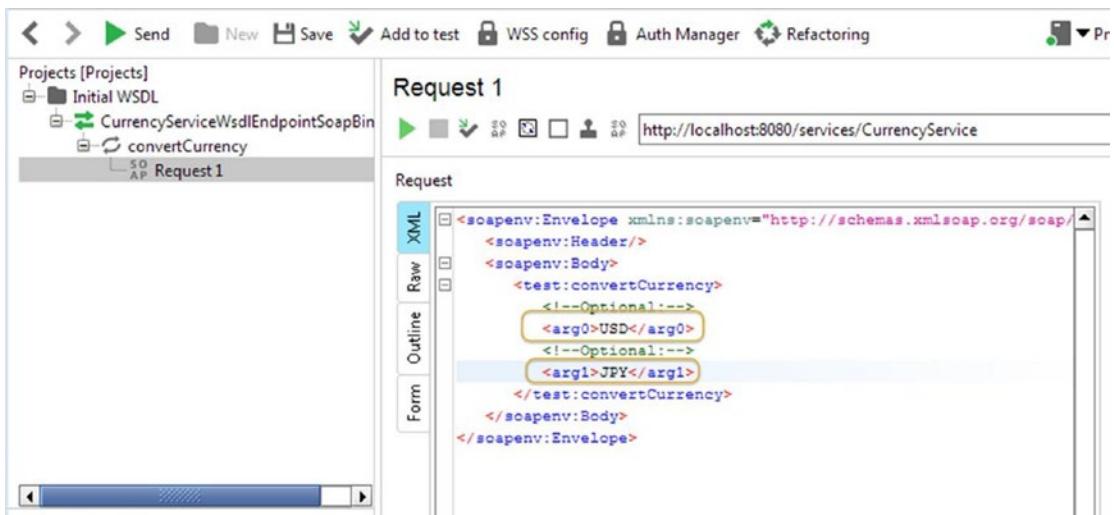


Figure 10-20. SOAP request

4. Click the green arrow to submit. You should see the response shown in Figure 10-21.



Figure 10-21. SOAP response

The WSDL for the Currency SOAP service is located at:

<http://localhost:8080/services/CurrencyService?WSDL>

You can view the WSDL using a browser (see Figure 10-22).



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="CurrencyService">
  <wsdl:types>
    <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://test.cxf.grails.org/" elementFormDefault="qualified">
      <xss:element name="convertCurrency" type="tns:convertCurrency"/>
      <xss:element name="convertCurrencyResponse" type="tns:convertCurrencyResponse"/>
    </xss:schema>
    <xss:complexType name="convertCurrency">
      <xss:sequence>
        <xss:element minOccurs="0" name="arg0" type="xs:string"/>
        <xss:element minOccurs="0" name="arg1" type="xs:string"/>
      </xss:sequence>
    </xss:complexType>
    <xss:complexType name="convertCurrencyResponse">
      <xss:sequence>
        <xss:element minOccurs="0" name="return" type="xs:string"/>
      </xss:sequence>
    </xss:complexType>
  </wsdl:types>
  <wsdl:message name="convertCurrencyRequest">
    <wsdl:part name="parameters" type="tns:convertCurrency"/>
  </wsdl:message>
  <wsdl:message name="convertCurrencyResponse">
    <wsdl:part name="result" type="tns:convertCurrencyResponse"/>
  </wsdl:message>
  <wsdl:operation name="convertCurrency">
    <wsdl:input message="convertCurrencyRequest"/>
    <wsdl:output message="convertCurrencyResponse"/>
  </wsdl:operation>
  <wsdl:portType name="CurrencyServicePortType">
    <wsdl:operation name="convertCurrency">
      <wsdl:input message="convertCurrencyRequest"/>
      <wsdl:output message="convertCurrencyResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CurrencyServiceSoapBinding" type="tns:CurrencyServicePortType">
    <wsdl:operation name="convertCurrency">
      <wsdl:input>
        <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
          <soap:Body>
            <ns1:convertCurrency xmlns:ns1="http://test.cxf.grails.org/">
              <arg0>USD</arg0>
              <arg1>JPY</arg1>
            </ns1:convertCurrency>
          </soap:Body>
        </soap:Envelope>
      </wsdl:input>
      <wsdl:output>
        <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
          <soap:Body>
            <ns1:convertCurrencyResponse xmlns:ns1="http://test.cxf.grails.org/">
              <return>121.12</return>
            </ns1:convertCurrencyResponse>
          </soap:Body>
        </soap:Envelope>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CurrencyService">
    <wsdl:port name="CurrencyServiceSoapPort" binding="tns:CurrencyServiceSoapBinding">
      <wsdl:address>http://localhost:8080/services/CurrencyService</wsdl:address>
    </wsdl:port>
  </wsdl:service>

```

Figure 10-22. WSDL for the Currency SOAP service

Let's see how to test SOAP using JMeter with an example.

Follow these steps or download SOAPTestPlan.jmx.¹⁷

1. Create a test plan and give it a meaningful name, such as SOAP Test Plan.
2. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group. Configure **Number of Threads (Users)** as 1 and **Loop Count** as 1000.

¹⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_10/SOAPTestPlan.jmx

- Click on Thread Group and go to Edit ▶ Add ▶ Configuration Element to add HTTP Header Manager. Configure **Headers Stored in the Header Manager**, **Name** as Content-Type and **Value** as text/xml (see Figure 10-23).

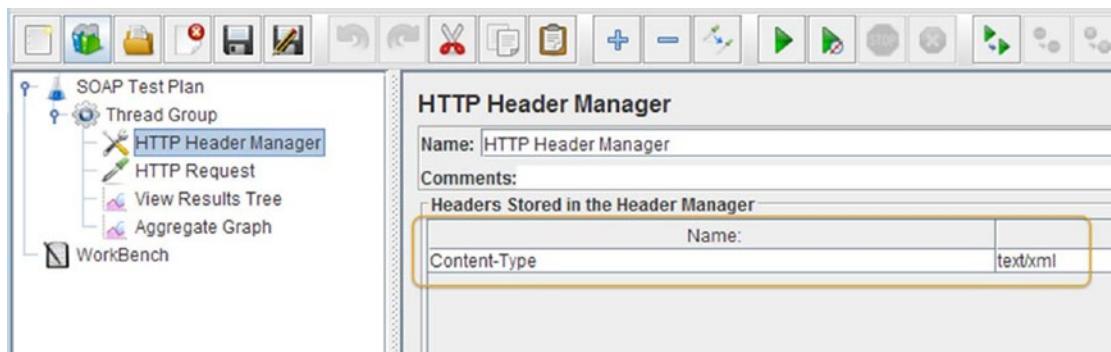


Figure 10-23. HTTP header manager

- Click on Thread Group and go to Edit ▶ Add ▶ Sampler. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and **Path** as /services/CurrencyService. Copy the SOAP Request XML, which was generated by the SoapUI earlier, into the body data area (see Figure 10-24).

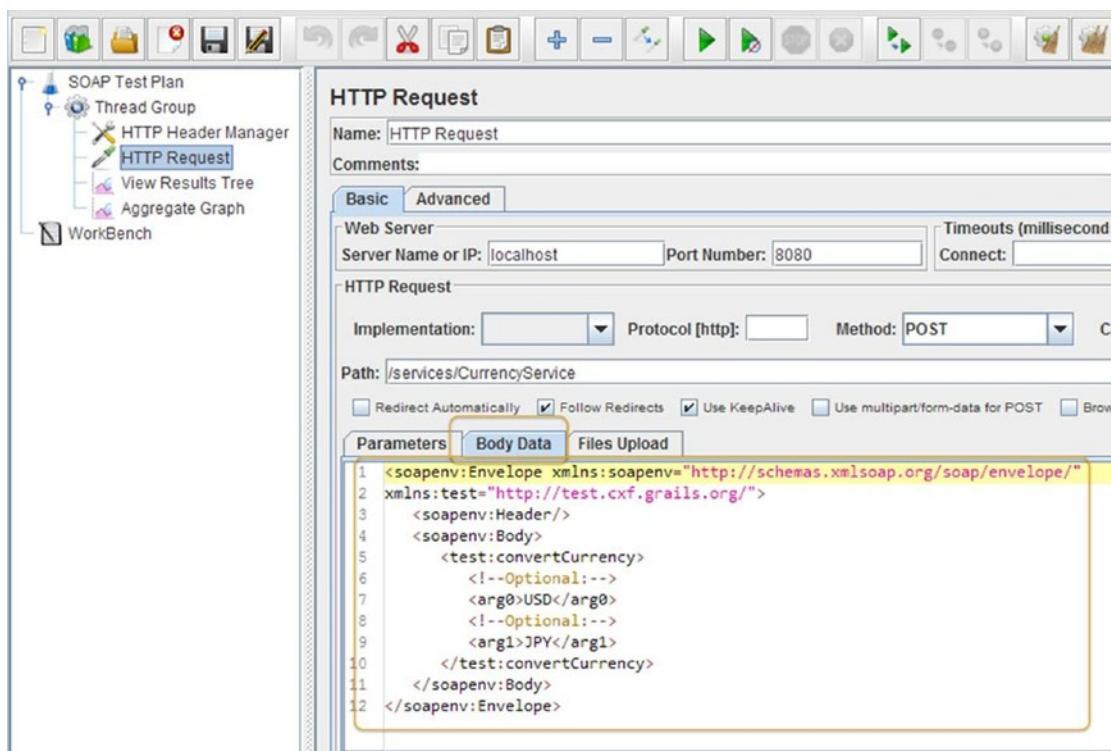
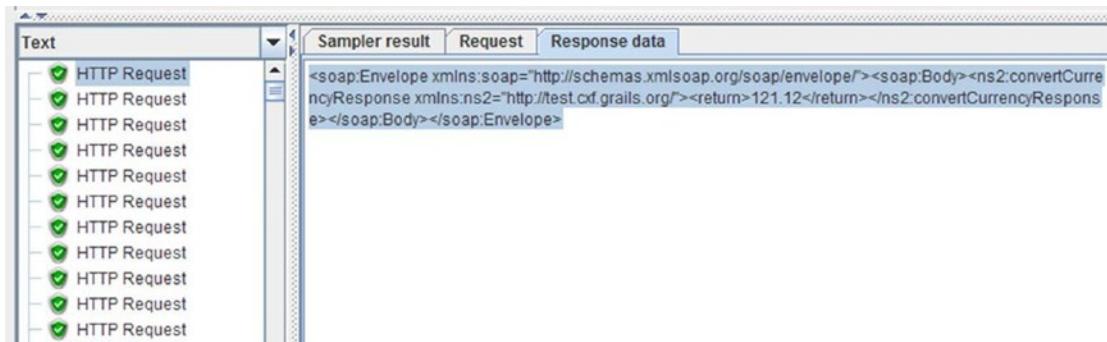


Figure 10-24. SOAP XML-HTTP request

5. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
6. Run the test.

The results are shown in Figure 10-25.



The screenshot shows the JMeter interface with the 'View Results Tree' listener selected. On the left, a tree view lists 15 'HTTP Request' entries, each with a green checkmark indicating success. On the right, the 'Response data' tab is active, displaying the XML structure of a SOAP response. The XML includes a soap:Envelope with a soap:Body containing ns2:convertCurrencyResponse, which returns the value 121.12. The XML structure is as follows:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:convertCurrencyResponse xmlns:ns2="http://test.cxf.grails.org/"><return>121.12</return></ns2:convertCurrencyResponse></soap:Body></soap:Envelope>
```

Figure 10-25. SOAP response XML format

Conclusion

In this chapter, we learned about the advanced features of JMeter. We learned about JDBC, FTP, and REST/JSON testing using their specific samplers and SOAP performance testing using the HTTP Request Sampler. We also learned about AJAX request performance testing as well as mobile web application performance testing. In the next chapter, we will go through a case study where we will be learning to apply JMeter skills that we learned in this and previous chapters to deal with a real-life performance testing project.

CHAPTER 11



Case Study: Digital Toys Inc.

This chapter presents a case study of a hypothetical e-commerce company called Digital Toys Inc. In the process, we walk you through various human, financial, and technical dynamics that revolve around web application performance.

At the end of this chapter, you will gain enough insights about performance test planning and execution to be of help in your projects. It is good to go through this chapter regardless of your previous experience, as it talks about performance testing the project.

The Need for Speed

Digital Toys Inc. was a pioneer in selling digital gadgets on the Internet. They became very popular and were very profitable for a long time. However, with new companies cropping up in the space, they were losing business, which caused concern to senior management.

In the company meeting, Nancy, their Chief Financial Officer, noted a trend of declining sales. The Customer Support Manager noted that the support calls had increased. A lot of customers complained about the web site speed; some of them complained that they were not able to complete their orders, requiring the support staff to place the orders instead. This caused increased wait times for the customers, further aggravating the situation.

Jay, their Product Manager, did not see any issues with the merchandise or pricing as nothing had changed, and it was similar to their competitors.

The management decided to engage Insight Management Consultants Inc. to solve this mystery and to guide the company on to the right track. They analyzed and submitted their findings and recommendations to the senior management.

Their report was elaborate and concluded that Digital Toys Inc. was losing business because of the web site being slow. They also quoted various industry sources in support of their argument.

"If an e-commerce site is making \$100,000 per day, a 1-second page delay could cost \$2.5 million in lost sales a year."

—Kissmetrics Blog¹

It was clear that they needed to improve the web site's performance. Martin, the CEO, entrusted this matter to the Director of Engineering, who in turn delegated it to Alex, one of their best engineers.

¹Source: Sean Work, Minister of Propaganda at Kissmetrics, <https://blog.kissmetrics.com/loading-time/>

Addressing the Problem

Alex read through the report prepared by Insight Management Consultants Inc.

Later, he met with the Customer Support Manager, the Product Manager, and the Marketing Manager. He created a list of the problem web pages and use-cases and prioritized them. He also looked at the web analytics report and the web access logs to understand the user access patterns of the Digital Toys Inc. web application.

Performance Goals

Alex obtained the response time of the competitors using Alexa.Com² and compared the results with WebPageTest.Org.³ He obtained additional performance metrics of the competitors from WebPageTest.Org. Alex also referred to the report prepared by Insight Management Consultants Inc. Putting it all together, Alex established the performance goals shown in Table 11-1.

Table 11-1. Performance Goals

	Minimum Response Time	Average Response Time	Maximum Response Time
Web pages	2 seconds	4 seconds	6 seconds
Use-cases	60 seconds	90 seconds	120 seconds
	Resource Utilization Percent		
CPU			60
Memory			50
Network			10

Tip Establish measurable performance goals.

Performance Test Specification

Alex selected the use-cases based on the feedback received from various sources, such as Insight Management Consultants Inc., the Customer Support Manager, the Product Manager, and the Marketing Manager. He captured sufficient detail for writing the test scripts.

²<http://www.alexa.com>

³<http://www.webpagetest.org>

He arrived at the use-cases described here.

1. User Lands on the Home Page and Browses the Products (Table 11-2)

Table 11-2. Steps for Landing Page

No.	Step
1	Home Page
2	Browse Products

2. User Registration (Table 11-3)

Table 11-3. Steps for User Registration

No.	Step
1	Home Page
2	Sign Up
3	Submit the Registration Form
4	Log Out

3. User Places an Order for the First Time (Table 11-4)

Table 11-4. Steps for Ordering the First Time

No.	Step
1	Home Page
2	Login Page
3	Index/Product Catalog Page
4	Details
5	Add To Cart
6	Check Out
7	Billing/Shipping Address
8	Credit Card Entry
9	Place Order
10	Order History
11	Sign Out

4. User Places an Order for the Second Time (Table 11-5)

Table 11-5. Steps for Ordering the Second Time

No.	Step
1	Home Page
2	Login Page
3	Index/Product Catalog Page
4	Details
5	Add To Cart
6	Check Out
7	Place Order
8	Order History
9	Sign Out

5. User Edits the Billing/Shipping Address (Table 11-6)

Table 11-6. Steps for Editing Billing/Shipping Address

No.	Step
1	Home Page
2	Login Page
3	Index/Product Catalog Page
4	Billing/Shipping Address
5	Sign Out

6. User Edits the Payment Information (Table 11-7)

Table 11-7. Steps for Editing Payment Information

No.	Step
1	Home Page
2	Login Page
3	Index/Product Catalog Page
4	Add Credit Card Details
5	Sign Out

7. User Uses the Continue Shopping Option (Table 11-8)

Table 11-8. Steps for Using the Continue Shopping Option

No.	Step
1	Home Page
2	Login Page
3	Index/Product Catalog Page
4	Details
5	Add To Cart
6	Continue Shopping
7	Check Out
8	Billing/Shipping Address
9	Credit Card Details
10	Place Order
11	Order History
12	Sign Out

Note Write the test cases in plain English. This can serve as a requirement for creating a performance test plan and it is easy to get feedback from other team members.

Tool Selection

Alex searched the Internet and found many performance testing tools. He shortlisted two tools that he felt would solve the problem. They needed the team to learn a proprietary scripting language.

Alex discussed these two tools with Bob, his manager. Bob appreciated Alex for his efforts but informed him that there was no budget. Further, the problem needed to be solved in a couple of days and there was no time for training. Bob asked Alex to be creative.

Alex discovered Apache JMeter.⁴ He soon found out that JMeter is a time-tested, robust tool for performance testing. JMeter does not require coding in any scripting language. It is an open source tool and it is accepted very well in the performance testing community.

Note Use a robust tool with good community support that meets your needs and has a short learning curve.

Test Environment

Later, Alex requested Bob for performance test infrastructure. Bob replied that it would take at least three weeks to procure new hardware and to provision it. Besides, they did not have the budget. Bob asked Alex to investigate if he could repurpose the existing functional testing infrastructure to set up the performance test environment.

Alex was happy that all he had to do was to install JMeter on these servers; everything was already there.

⁴<http://jmeter.apache.org/>

Test Data Preparation

Alex asked Bob for a copy of production data to set up a test database. Bob replied that it is not possible due to security reasons and asked Alex to create a test data set.

Alex obtained special clearance from the Chief Information Officer and used the production data but obfuscated the critical and personally identifying data. He created the test data for performance testing. This technique ensured that the performance data resembled the production data without any risk of exposure of critical data like credit card information, e-mails, or names. For the product catalog, Alex used an in-house loader to load product information.

Tip In the test environment, obfuscate the production data to prevent exposing critical data like credit card information and other personal data.

User Load Pattern

Alex obtained the web access logs and identified the URL patterns and a set of frequently accessed URLs.

He collaborated with the Database Administrator and obtained the following:

- The count of active users connecting to the production database servers
- The count of users who placed an order and the order count
- The number of times that the address/payment information got modified
- The count of customers who chose to continue shopping

Alex asked the database administrator for a set of SQL statements that he could run on the database for getting the count of users for different scenarios. He ran these scripts on the test database and found out numbers. He then analyzed web server access logs for the URL patterns based on various scenarios and compared them with the database counts to verify if the information was correct. After confirmation, he incorporated the data into Table 11-9.

Table 11-9. User Load Pattern

Variable	Result
The count of users who are logged in at one point in time.	100,000
The use-case and the percentage of users.	Browsing catalog: 10% Ordering first time: 60% Ordering second time: 10% Changing address: 5% Changing payment information: 5% Continue shopping: 10%
Time of day and duration.	Most of the users use the application during early morning and at the end of the day. The duration of load is between 1 to 2 hours.
Critical days of application usage.	Five days prior to Christmas.

Tip Express the usage in terms of percentages. This makes it easier to perform the test with various counts of simulated users.

Application Build

Alex used the mock payment gateway, which was originally intended for functional testing. Bob provided him with a custom build that integrated the mock payment gateway to avoid using the production gateway.

Alex set up the application server with the custom build and the database server with obfuscated production data. He took a backup of the application build and the database dump to utilize it to refresh the testing environment every time a fresh round of performance testing was needed.

Tip Use mock services of third-party applications while doing performance testing.

Using JMeter

Before starting with the test script development, Alex went through the JMeter component definitions.

- *Test Plan*: Every JMeter test script contains a test plan as a root node. It contains a thread group, which contains one or more of the following child nodes:
Sampler, Logical Controller, Listener, Assertions, Timer, Config Element
- *Thread Group*: This is synonymous to a web user.
- *Samplers*: Send requests to the server. JMeter ships with many samplers out of which the *HTTP Request Sampler* is used most often. It is used to simulate a web request.
- *Logical Controllers*: Provide common programming constructs that provide control flow and are used to structure JMeter test scripts. Using controllers, you can specify the order in which the samplers are processed.
- *Listeners*: Used to display the results of the server responses generated due to sampler requests. They are processed at the end of the scope in which they are found.
- *Assertions*: Used to validate the server responses generated due to sampler requests. For example, to verify that for a HTTP request, the HTTP Response Code is 200.
- *Timers*: Used to insert a delay before the sampler is executed. These are used to simulate the user think time before the next action. They are processed before each sampler in the scope in which they are found. *Timers* are only processed in conjunction with a sampler.
- *Config Elements*: Used to set up defaults and variables for later use by other components. They are processed at the start of the scope in which they are found.

Alex started with the development of JMeter test scripts based on the use-cases. He followed an incremental approach and learned along the way.

Test Script Development

Use-Case # 1 User Lands on the Home Page and Browses the Products.

Alex followed these steps to create a test plan.

1. Create a test plan and give it a meaningful name, such as **User browsing products on Home Page**.
2. Click on Test Plan and go to **Edit > Add > Threads (Users)**. Add Thread Group.
3. Click on Thread Group and go to **Edit > Add > Sampler**. Add HTTP Request. Configure **Server Name or IP** as localhost, **Port Number** as 8080, and path as /dt.
4. Click on Thread Group and go to **Edit > Add > Listener**. Add View Results Tree.
5. Run the test.

The results are shown in Figure 11-1.

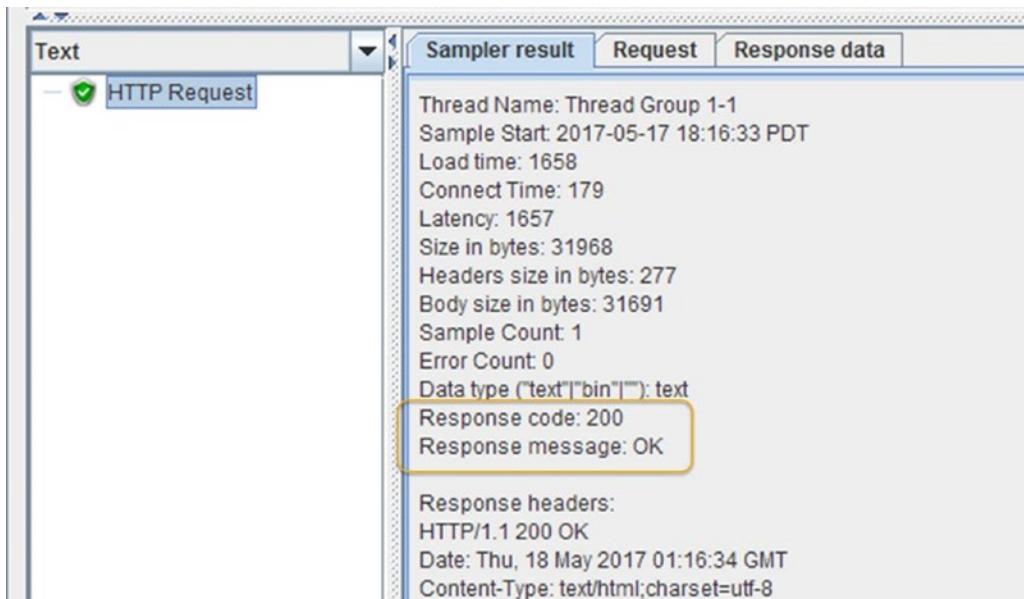


Figure 11-1. Home page view results tree

6. Save the test plan as **DTHomePageTestPlan.jmx**.⁵

⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTHomePageTestPlan.jmx

Use-Case # 2 User Registration.

As Alex walked through the steps manually, he realized that the process involved filling out forms and posting the form data to the server. He found it to be more complex compared to the previous use-case, where obtaining data from the server using a GET request was easy.

Alex discovered that he could record the manual steps using JMeter. He had to first configure the proxy and then record the test.

Alex followed these steps to configure the proxy settings in the Firefox browser.

1. Open the Firefox browser and go to Preferences ➤ Advanced ➤ Network ➤ Connection Settings ➤ Manual Proxy Configuration. Configure **HTTP Proxy** as **localhost** and **Port** as **7070**.
2. Select Use This Proxy Server for All Protocols.
3. Clear the No Proxy For text area.
4. Click OK.

Now, for the JMeter test script, Alex followed these steps.

1. In JMeter, create a test plan and give it a meaningful name, such as **User Registration**.
2. Click on WorkBench and go to Edit ➤ Add ➤ Non-Test Elements. Add **HTTP(S) Script Test Recorder**.
3. In the Global Settings, configure **Port** as **7070**.
4. Select **Target Controller** as **WorkBench ➤ HTTP(S) Test Script Recorder**.
5. Click on the Start button to start recording.
6. In the Firefox browser, perform the steps of the use-case called **User Registration**.

You will see the recording in JMeter, as shown in Figure 11-2.

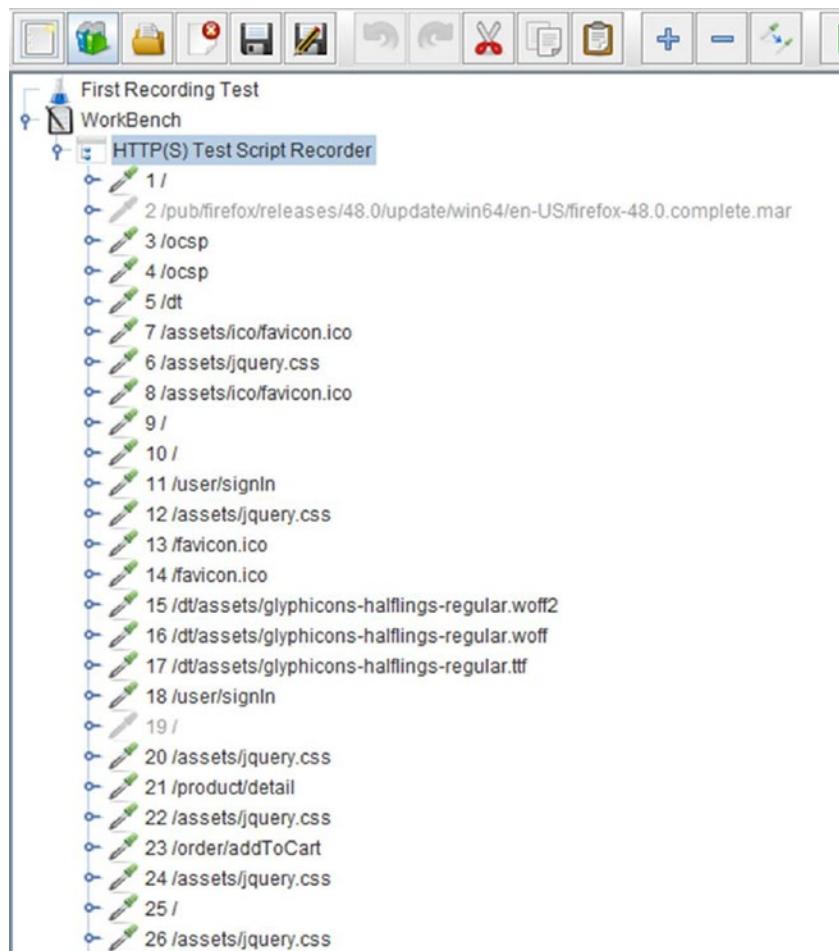


Figure 11-2. Recording the user registration

In these results, the recorded steps have requests for .bmp, .css, .js, .git, .ico, .jpeg, .png, .swf, .woff, .woff2, and .ttf files, which are not required. Alex wanted to exclude these requests. He decided to delete the previous recording and re-record the test. You would follow these steps to do so:

1. In JMeter, click on the Stop button.
2. To delete the recording, select all the recorded browser requests and go to Edit ► Remove. Confirm the dialog box.

3. Exclude these regular expressions (see Figure 11-3):
`.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*`

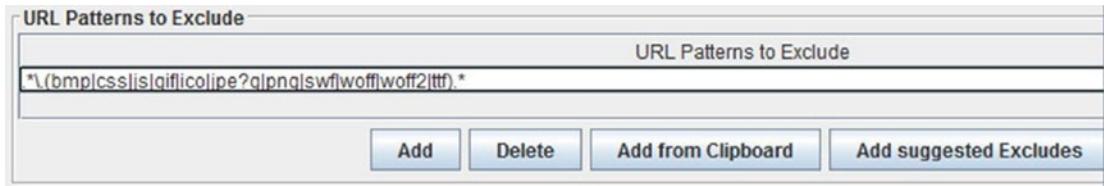


Figure 11-3. Exclusion of regular expressions

4. Click on the Start button.
5. In the Firefox browser, perform the steps of the use-case called User Registration.
6. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
7. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
8. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
9. Save the test plan as DTUserRegistrationTestPlan.jmx.⁶

Alex now understood the process of creating a test script by recording browser activity. He now had a test script that performed user registration. Next, Alex wanted to simulate the load of hundreds of users.

Alex learned about a configuration element called *CSV Data Set Config*, which is used to load data from a file in .csv format. He planned to use it to load the user details for the user registration form.

Alex followed these additional steps.

1. Click on the user registration HTTP request POST request and go to Edit ▶ Add ▶ Config Element. Add CSV Data Set Config.

⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTUserRegistrationTestPlan.jmx

2. Configure CSV Data Set Config, as shown in Figure 11-4.

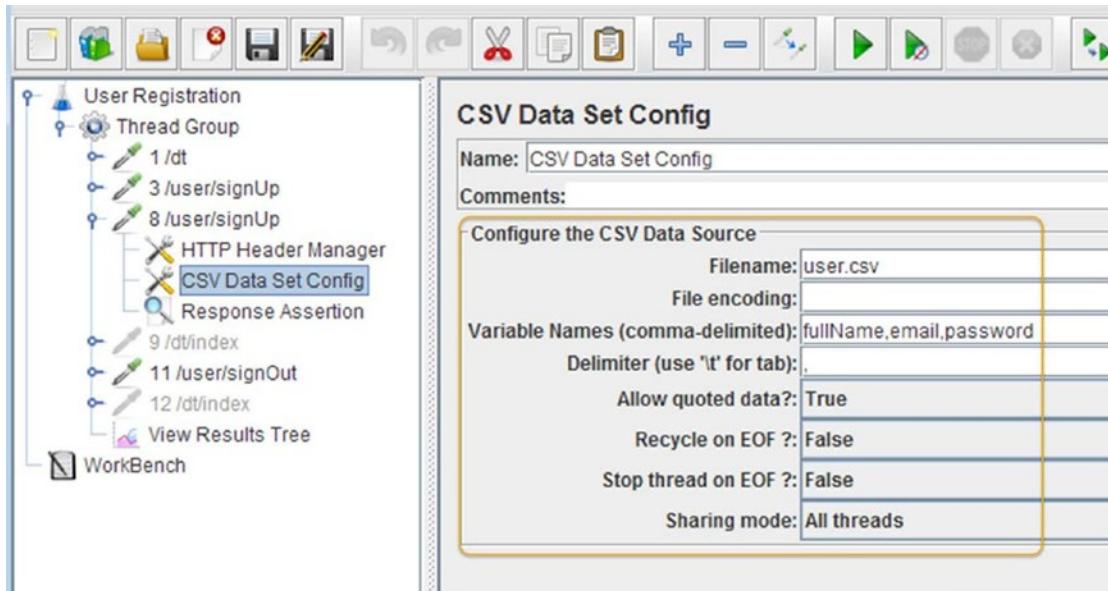


Figure 11-4. User CSV configuration

3. Save the test plan.

Create the user.csv file containing the user details organized into three columns: fullName, email, and password. Place the file in the same directory as the test script.

```
Jagdeep,jagdeep@gmail.com,1234
Sai,sai@gmail.com,1234
Ganesh,ganesh@gmail.com,1234
Gopal,gopal@gmail.com,1234
```

Alex now learned the process of parameterization in JMeter by passing the values from an external CSV file. He followed the additional steps to configure the test plan to register the users listed in the user.csv file.

1. Click on Thread Group and configure the **Number of Threads (Users)** as equal to the number of rows in the user.csv file.
2. Save the test plan.
3. Run the test.

After the test run, Alex noted that the users in the user.csv file were generated successfully. By using the username/password to manually log in to the Digital Toys Inc. web application, he verified that the order was created.

Use-Case # 3 User Places an Order for the First Time.

Recording the browser actions is one of the best features of JMeter, and it helped Alex create the test scripts easily. He configured the browser and JMeter to listen on the same port and followed these steps to record another use-case.

1. Create a test plan and give it a meaningful name, such as **User Placing an Order for the First Time**.
2. Click on WorkBench and go to Edit ▶ Add ▶ Non-Test Elements and add **HTTP(S) Script Test Recorder**.
3. In the Global Settings, configure **Port** as **7070**.
4. Select **Target Controller** as **WorkBench > HTTP(S) Test Script Recorder**.
5. Exclude these regular expressions:
`.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*`
6. Click on the Start button.
7. In the Firefox browser, perform the steps of the use-case called **User Placing an Order for the First Time**.
8. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
9. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of *Thread Group*.
10. Click on *Thread Group* and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
11. Save the test plan as **DTFirstOrderTestPlan.jmx**.⁷
12. Run the test.

Alex noticed that the order was not created in the web application. He quickly found the reason for this—cookies! The web application was using cookies for the session management but JMeter was not preserving the cookies between requests. He decided to add **HTTP Cookie Manager** and followed these additional steps.

1. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add **HTTP Cookie Manager**.
2. Run the test again.

Alex was able to verify that the new order was created, and it was visible on the web application under order history.

After the test plan was developed and tested, Alex reviewed it to see if anything could be optimized. He realized that the Server Name or IP and Port Number options were repeated. He decided to capture the repeated information in **HTTP Request Defaults**, so he followed these additional steps.

1. Click on Thread Group and go to Edit ▶ Add ▶ Config Element and add **HTTP Request Defaults**. Configure **Server Name or IP** as **localhost** and **Port Number** as **8080**.
2. Remove the values of **Server Name or IP** and **Port Number** from each HTTP request.
3. Save the test plan.

⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

■ **Tip** Use HTTP Request Defaults and keep the server configuration in one place to make the test plan portable. If the test environment changes, it is an easy matter to change it, as it needs to be updated just in one place.

Use-Case # 4 User Places an Order for the Second Time.

For the next use-case, Alex chose to use the *HTTP(S) Test Script Recorder*. In addition, he knew that the test plan should contain the *HTTP Cookie Manager* and *HTTP Request Defaults* samplers.

He configured the browser and JMeter to listen on the same port and followed these steps to record another use-case.

1. Create a test plan and give it a meaningful name, such as **User Placing an Order for the Second Time**.
2. Click on WorkBench and go to Edit ► Add ► Non-Test Elements. Add **HTTP(S) Script Test Recorder**.
3. In the Global Settings, configure **Port** as **7070**.
4. Select **Target Controller** as **WorkBench > HTTP(S) Test Script Recorder**.
5. Exclude these regular expressions:
.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*
6. Click on the Start button.
7. In the Firefox browser, perform the steps of the use-case called **User Placing an Order for the Second Time**.
8. Click on Test Plan and go to Edit ► Add ► Threads (Users). Add Thread Group.
9. Select all the recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
10. Click on Thread Group and go to Edit ► Add ► Config Element. Add **HTTP Cookie Manager**.
11. Click on Thread Group and go to Edit ► Add ► Config Element. Add **HTTP Request Defaults**. Configure **Server Name or IP** as **localhost** and **Port Number** as **8080**.
12. Remove the values of **Server Name or IP** and **Port Number** from each HTTP request.
13. Click on Thread Group and go to Edit ► Add ► Listener. Add **View Results Tree**.
14. Save the test plan as **DTSecondTimeOrderTestPlan.jmx**.⁸
15. Run the test.

Alex was able to verify that the new order was created, and it was visible in the web application under order history.

⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTSecondTimeOrderTestPlan.jmx

Use-Case # 5 User Edits the Billing/Shipping Address.

Alex configured the browser and JMeter to listen on the same port and followed these steps to record another use-case.

1. Create a test plan and give it a meaningful name, such as User Editing Billing/Shipping Address.
2. Click on WorkBench and go to Edit ► Add ► Non-Test Elements. Add HTTP(S) Script Test Recorder.
3. In the Global Settings, configure **Port** as 7070.
4. Select **Target Controller** as WorkBench > HTTP(S) Test Script Recorder.
5. Exclude these regular expressions:
.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*
6. Click on the Start button.
7. In the Firefox browser, perform the steps of the use-case entitled User Editing Billing/Shipping Address.
8. Click on Test Plan and go to Edit ► Add ► Threads (Users). Add Thread Group.
9. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
10. Click on Thread Group and go to Edit ► Add ► Config Element. Add HTTP Cookie Manager.
11. Click on Thread Group and go to Edit ► Add ► Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
12. Remove values of **Server Name or IP** and **Port Number** from each HTTP request.
13. Click on Thread Group and go to Edit ► Add ► Listener. Add View Results Tree.
14. Save the test as DTChangeAddressTestPlan.jmx.⁹
15. Run the test.

Alex was able to verify that the old address was updated.

Use-Case # 6 User Edits the Payment Information.

Alex configured the browser and JMeter to listen on the same port and followed these steps to record another use-case.

1. Create a test plan and give it a meaningful name, such as User Editing Payment Information.
2. Click on WorkBench and go to Edit ► Add ► Non-Test Elements. Add HTTP(S) Script Test Recorder.
3. In the Global Settings, configure **Port** as 7070.
4. Select **Target Controller** as WorkBench > HTTP(S) Test Script Recorder.
5. Exclude these regular expressions:
.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*

⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

6. Click on the Start button.
7. In the Firefox browser, perform the steps of the use-case called **User Editing Payment Information**.
8. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
9. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
10. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.
11. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
12. Remove the values of **Server Name or IP** and **Port Number** from each HTTP request.
13. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
14. Save the test as DTChangePaymentInfoTestPlan.jmx.¹⁰
15. Run the test.

Alex was able to verify that the old payment information was updated.

Use-Case # 7 User Uses the Continue Shopping Option.

Alex configured the browser and JMeter to listen on the same port and followed these steps to record another use-case.

1. Create a test plan and give it a meaningful name, such as **User Using Continue Shopping Option**.
2. Click on WorkBench and go to Edit ▶ Add ▶ Non-Test Elements. Add HTTP(S) Script Test Recorder.
3. In the Global Settings, configure **Port** as 7070.
4. Select **Target Controller** as WorkBench > HTTP(S) Test Script Recorder.
5. Exclude the regular expression as:
.*\.(bmp|css|js|gif|ico|jpe?g|png|swf|woff|woff2|ttf).*
6. Click on the Start button.
7. In the Firefox browser, perform the steps of the use-case called **User Using Continue Shopping Option**.
8. Click on Test Plan and go to Edit ▶ Add ▶ Threads (Users). Add Thread Group.
9. Select all recorded browser actions from WorkBench, and then drag and add them as child elements of Thread Group.
10. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Cookie Manager.

¹⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

11. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add HTTP Request Defaults. Configure **Server Name or IP** as localhost and **Port Number** as 8080.
12. Remove the values of **Server Name or IP** and **Port Number** from each HTTP request.
13. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add View Results Tree.
14. Save the test as DTContinueShoppingTestPlan.jmx.¹¹
15. Run the test.

Alex was able to verify that the new order with multiple products was created, and it was visible in the web application under order history.

Validation of Test Steps

Alex decided to make sure that every test had *assertions* so that test execution could be validated. There were a couple of assertions at his disposal. Since JMeter assertions come with a runtime performance cost, he decided to limit them to response assertion and size assertion.

Use-Case # 1 User Lands on the Home Page and Browses the Products.

Alex followed these steps to update the existing test plan.

1. Open DTHomePageTestPlan.jmx.¹²
2. Click on HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
3. Run the test.

The results indicated that the test passed successfully.

To make sure that the test assertions were working as expected, Alex updated the Patterns To Test field to 500. After running the test, the test results indicated that the test failed; this confirmed that the assertions were working as expected (see Figure 11-5).

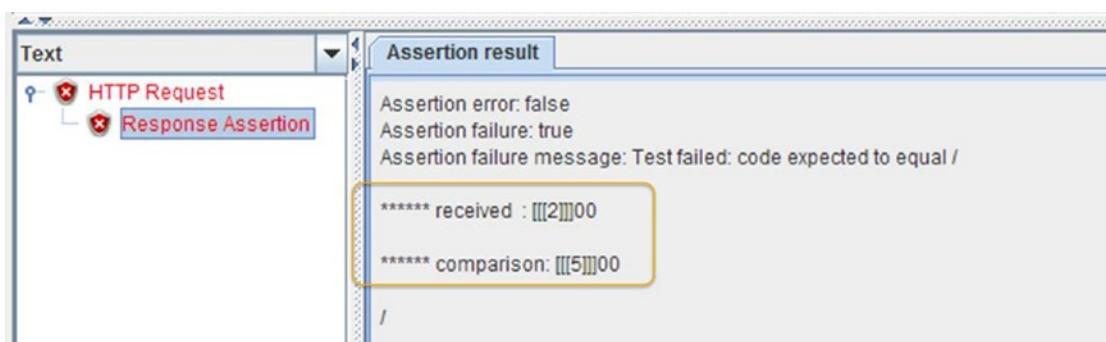


Figure 11-5. Assertion failure home page

¹¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

¹²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTHomePageTestPlan.jmx

Alex change 500 back to 200 and saved the updated test plan. The test was then successful.

■ Tip Use assertions to validate test results.

Use-Case # 2 User Registration.

Unlike prior test plans, this use-case involved many HTTP requests. Since assertions are costly, Alex added assertions only for important HTTP requests that performed POST request calls.

Alex followed these steps to update the existing test plan.

1. Open DTUserRegistrationTestPlan.jmx.¹³
2. Click on the second /user/signUp HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
3. Save the updated test plan.
4. Run the test.

The results indicated that the test passed successfully.

Use-Case # 3 User Places an Order for the First Time.

A typical web application has multiple HTTP request-response sequences, and Digital Toys Inc. web application is no different. Alex started with the next test plan. To make it more efficient, he decided to add assertions only for POST requests.

He decided to focus on /user/signIn, /order/addToCart, /user/addAddress, /user/addCard, and /user/signOut HTTP Requests.

■ Caution Login and subsequent logout are required to make sure that the session is closed. Otherwise, it may impact the overall performance results.

Alex followed these steps and updated the existing test plan.

1. Open DTFirstOrderTestPlan.jmx.¹⁴
2. Click on /user/signIn HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
3. Click on /order/addToCart HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
4. Click on /user/addAddress HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.

¹³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTUserRegistrationTestPlan.jmx

¹⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

5. Click on /user/addCard HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on /user/signOut HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
7. Save the updated test plan.
8. Run the test.

The results indicated that the test passed successfully.

Use-Case # 4 User Places an Order for the Second Time.

The next test plan also had multiple HTTP request-response sequences. Alex decided to focus on /user/signIn, /order/addToCart, and /user/signOut POST requests and added assertions.

Alex followed these steps to update the existing test plan.

1. Open DTSecondTimeOrderTestPlan.jmx.¹⁵
2. Click on /user/signIn HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
3. Click on /order/addToCart HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, Pattern Matching Rules as Equals, and **Patterns To Test** as 200.
4. Click on /user/signOut HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Save the updated test plan.
6. Run the test.

The results indicated that the test passed successfully.

Use-Case # 5 User Edits the Billing/Shipping Address.

This test plan also had multiple HTTP request-response sequences. Alex decided to focus on /user/signIn, /order/addAddress, and /user/signOut POST requests and added assertions.

Alex followed these steps to update the existing test plan.

1. Open DTChangeAddressTestPlan.jmx.¹⁶
2. Click on /user/signIn HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.

¹⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTSecondTimeOrderTestPlan.jmx

¹⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

3. Click on /order/addAddress HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
4. Click on /user/signOut HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Save the updated test plan.
6. Run the test.

The results indicated that the test passed successfully.

Use-Case # 6 User Edits Payment Information.

This test plan also has multiple HTTP request-response sequences. Alex decided to focus on /user/signIn, /order/addCard, and /user/signOut POST requests and added assertions.

Alex followed these steps to update the existing test plan.

1. Open DTChangePaymentInfoTestPlan.jmx.¹⁷
2. Click on /user/signIn HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
3. Click on /order/addAddress HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
4. Click on /user/signOut HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Save the updated test plan.
6. Run the test.

The results indicated that the test passed successfully.

Use-Case # 7 User Uses the Continue Shopping Option.

This test plan also had multiple HTTP request-response sequences. Alex decided to focus on /user/signIn, /order/addToCart, /user/addAddress, /order/addCard, and /user/signOut POST requests and added assertions.

Alex followed these steps to update the existing test plan.

1. Open DTContinueShoppingTestPlan.jmx.¹⁸
2. Click on /user/signIn HTTP Request and go to Edit ▶ Add ▶ Assertions. Add Response Assertion. Configure Apply to as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.

¹⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

¹⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

3. Click on /order/addToCart HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
4. Click on /user/addAddress HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
5. Click on /user/addCard HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
6. Click on /user/signOut HTTP Request and go to Edit ► Add ► Assertions. Add Response Assertion. Configure **Apply to** as Main Sample Only, **Response Field To Test** as Response Code, **Pattern Matching Rules** as Equals, and **Patterns To Test** as 200.
7. Save the updated test plan.
8. Run the test.

The results indicated that the test passed successfully.

Passing Variables Between Samplers

Alex reviewed all the test plans before running tests with multiple users.

For the Edit Address test plan, Alex needed to grab the value of addressId from the response of /user/addAddress GET HTTP request and pass it to /user/addAddress POST HTTP Request.

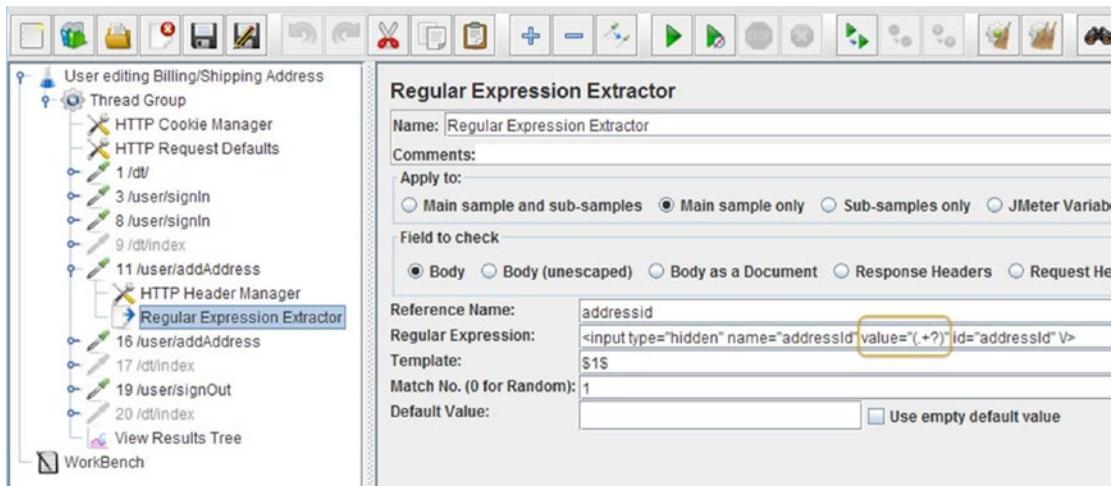
Similarly, for the Edit Payment Information test plan, he needed to grab the value of cardId from the response of /user/addCard GET HTTP request and pass it to /user/addCard POST HTTP request.

Alex followed these steps to update the existing test plan.

Updating Change Address Test Plan.

1. Open DTChangeAddressTestPlan.jmx.¹⁹
2. Click on /user/AddCard HTTP Request and go to Edit ► Add ► Post Processors. Add Regular Expression Extractor. Configure **Reference Name** as addressid, **Regular Expression** as <input type="hidden" name="addressId" value="(.+?) " id="addressId" \> that he found from the HTML code, **Template** as \$1\$, and **Match No. (0 for Random)** as 1 (see Figure 11-6).

¹⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

**Figure 11-6.** Address ID regular expression extractor

- Pass the reference name to the next HTTP request, as shown in Figure 11-7.

Send Parameters With the Request:	
Name:	Value
addressId	\$(addressid)
line1	Changed2AddressLine2
line2	Changed2AddressLine2

Figure 11-7. Pass the address ID to the next HTTP request

- Save the updated test plan.

Updating Change Payment Information Test Plan.

- Open DTChangePaymentInfoTestPlan.jmx.²⁰
- Click on /user/addCard HTTP Request and go to Edit ▶ Add ▶ Post Processors. Add Regular Expression Extractor. Configure **Reference Name** as **cardid**, **Regular Expression** as `<input type="hidden" name="addressId" value="(.+?)" id="addressId" \>` that he found from the HTML code, **Template** as `1`, and **Match No. (0 for Random)** as 1 (see Figure 11-8).

²⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

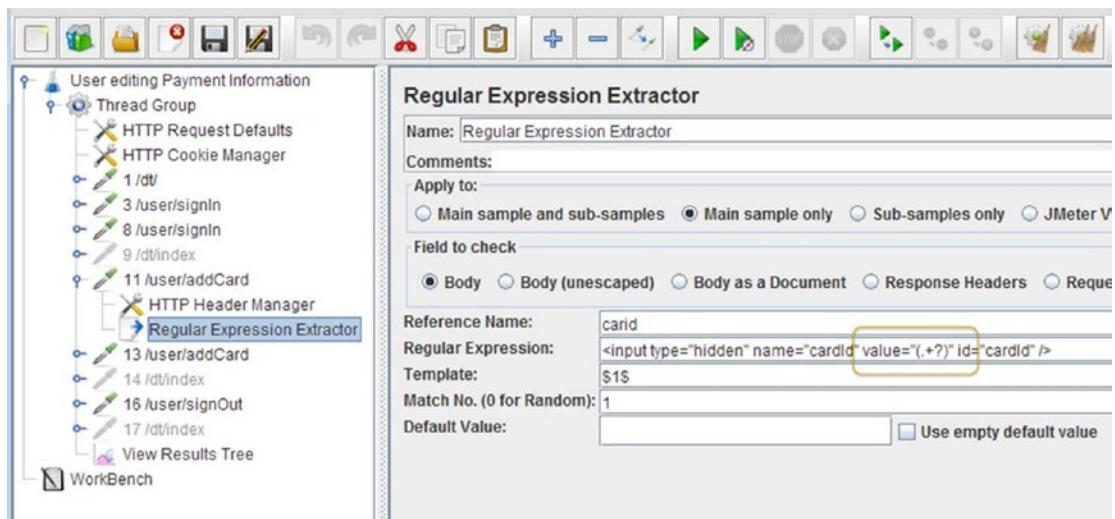


Figure 11-8. Card ID regular expression extractor

- Pass the reference name to the next HTTP request, as shown in Figure 11-9.

Parameters		Body Data	Files Upload
Send Parameters With the Request:			
Name:			Value
cardId		\${cardid}	
nameOnCard		User1	

Figure 11-9. Pass the card ID to the next HTTP request

- Save the updated test plan.

Running Tests with Multiple Users

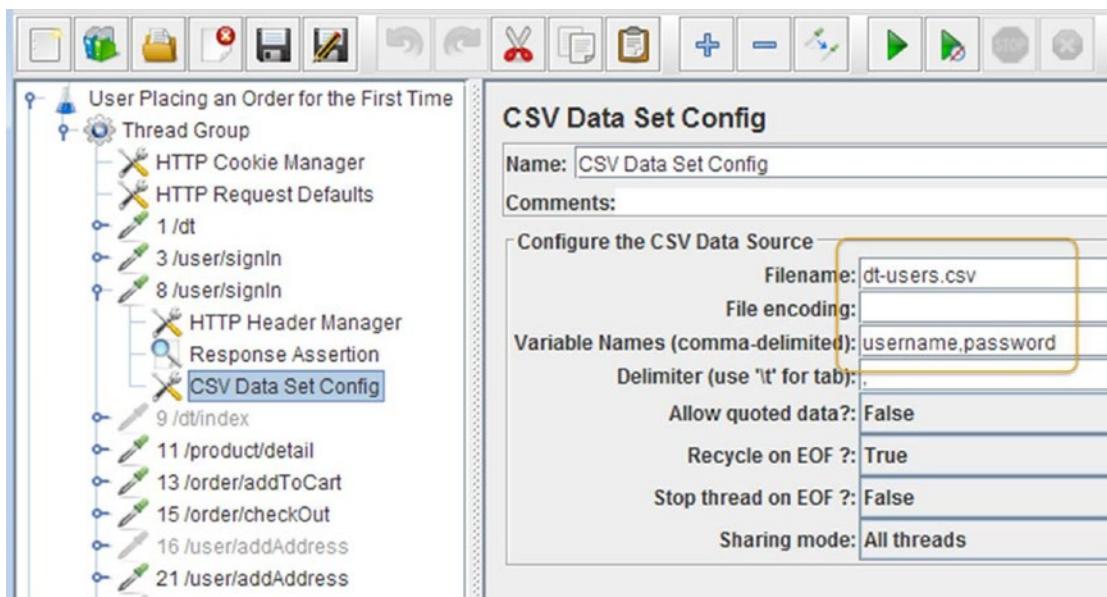
To simulate the load of multiple users accessing the application concurrently, Alex made a few changes to the existing test plans.

Required changes to the test plans are as follows:

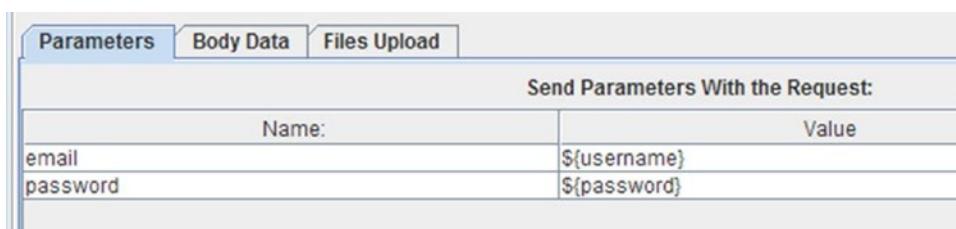
- Add *CSV Data Set Config*.
- In Configure the CSV Data Source, set the filename to the user list CSV file.
- In /user/signIn HTTP Request, make *CSV Data Set Config* the child element.
- Update Thread Group configuration with the number of threads/users.

Updating Case # 3: User Places an Order for the First Time.

1. Open DTFirstOrderTestPlan.jmx.²¹
2. Click on the second /user/signIn HTTP Request and go to Edit ▶ Add ▶ Config Element. Add CSV Data Set Config. Configure **Filename** as dtusers.csv, **Variable Names (comma-delimited)** as username,password, and **Delimiter (use \t for tab)** as "," (see Figure 11-10).

**Figure 11-10.** CSV data set config parameters

3. Click on the same /user/signIn HTTP request and configure parameters in the request transaction (see Figure 11-11).

**Figure 11-11.** Request parameters

4. Save the updated test plan.

²¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

Alex updated the remaining test plans by opening each plan one by one and following the previous steps.

- DTSecondTimeOrderTestPlan.jmx
- DTChangeAddressTestPlan.jmx
- DTChangePaymentInfoTestPlan.jmx
- DTContinueShoppingTestPlan.jmx

To simulate the load of 10 concurrent users, Alex ensured that the `dt-users.csv` file mentioned in the CSV *Data Set Config* had 10 users.

The next step was to configure Thread Group with the number of concurrent users and run the test.

Use-Case # 1 User Lands on the Home Page and Browses the Products.

1. Open `DTHomePageTestPlan.jmx`.²²
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

The results are shown in Figure 11-12.

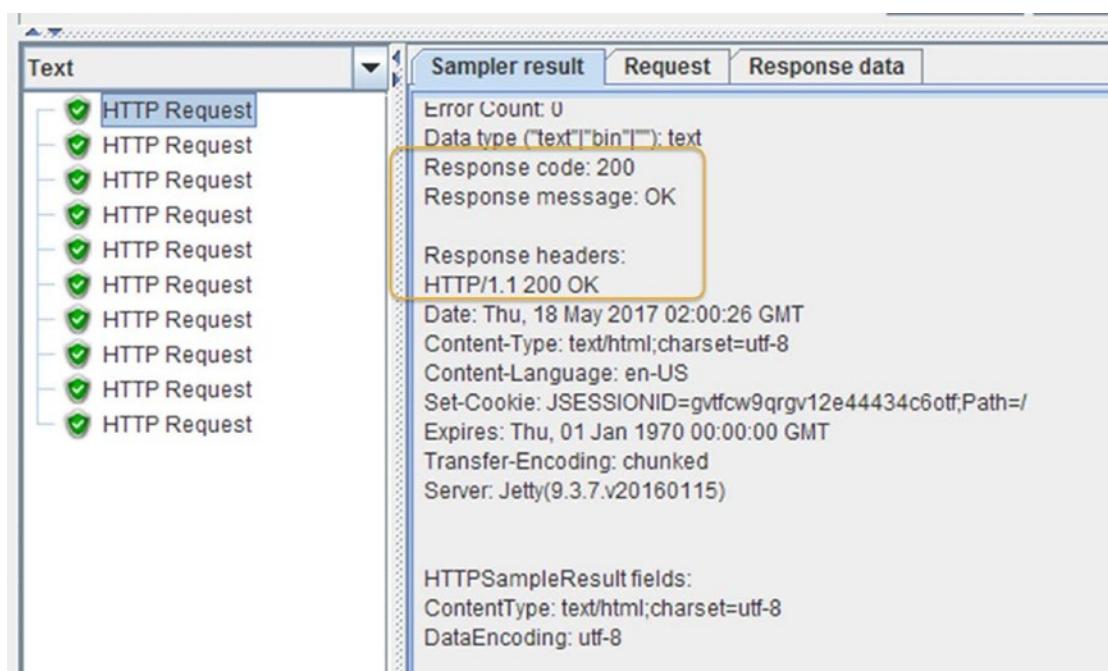


Figure 11-12. Landing page 10 threads

²²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTHomePageTestPlan.jmx

Use-Case # 3 User Places an Order for the First Time.

1. Open DTFirstOrderTestPlan.jmx.²³
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

The results are shown in Figure 11-13.

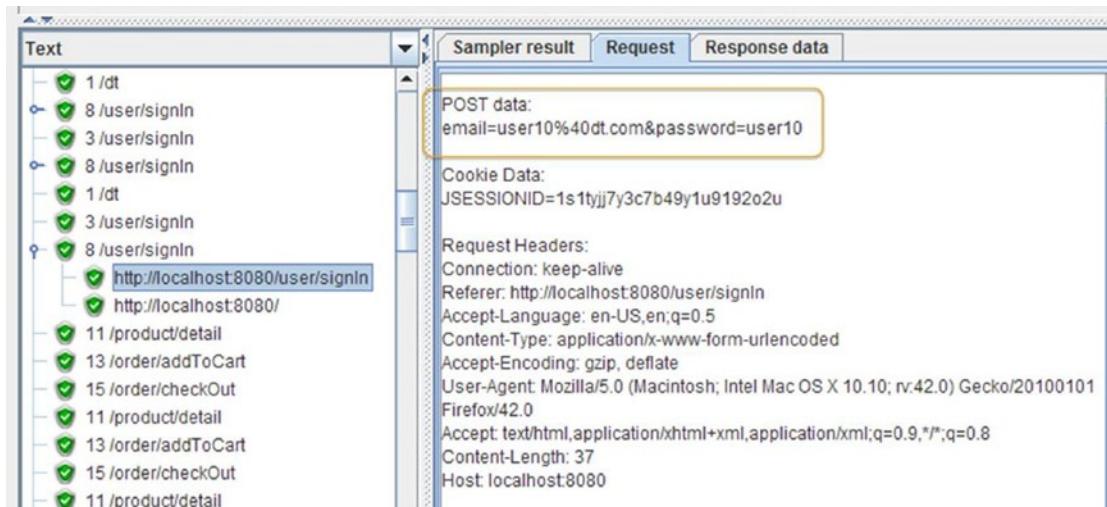


Figure 11-13. First order 10 threads

Use-Case # 4 User Places an Order for the Second Time.

1. Open DTSecondTimeOrderTestPlan.jmx.²⁴
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

²³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

²⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTSecondTimeOrderTestPlan.jmx

The results are shown in Figure 11-14.

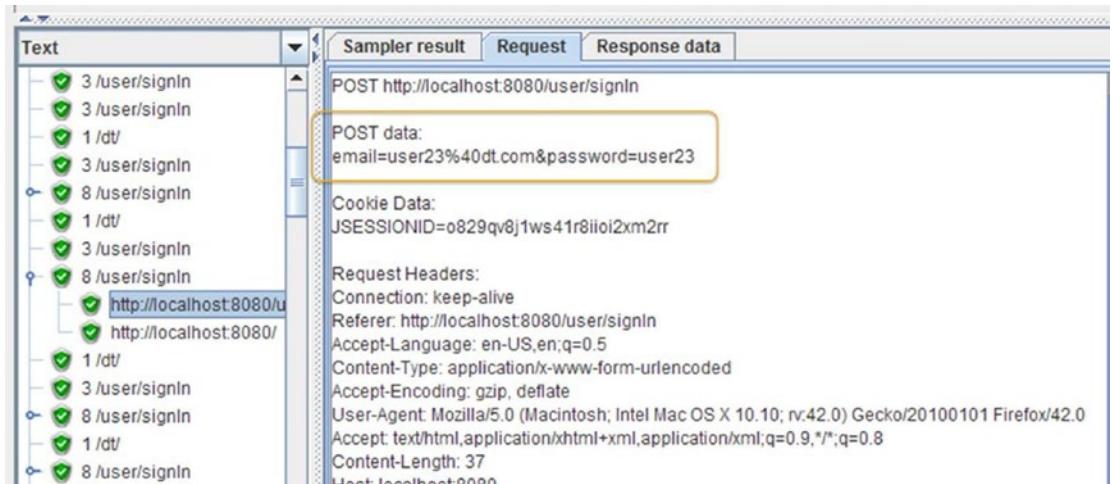


Figure 11-14. Second time order 10 threads

Use-Case # 5 User Edits the Billing/Shipping Address.

1. Open DTChangeAddressTestPlan.jmx.²⁵
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

²⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

The results are shown in Figure 11-15.

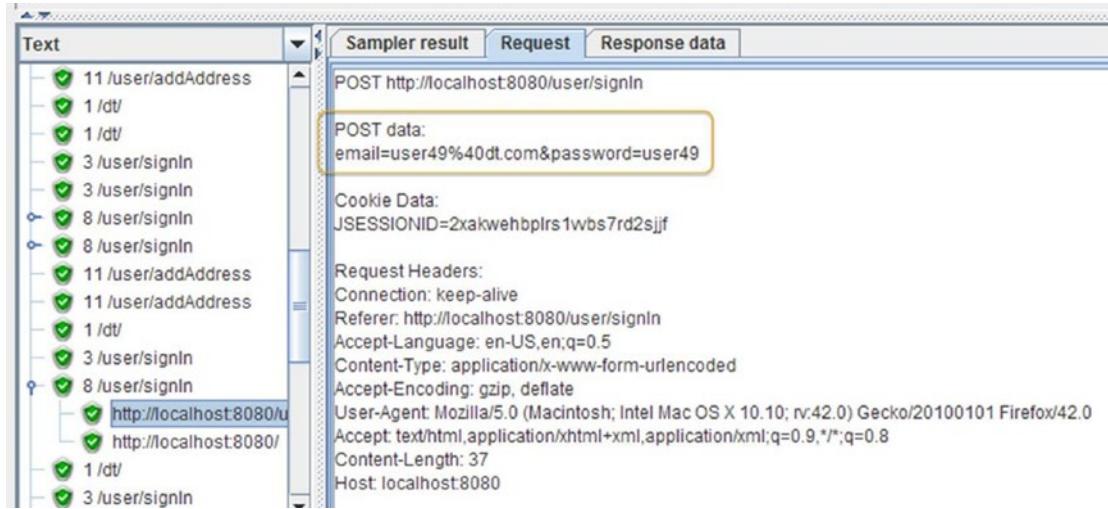


Figure 11-15. Change address 10 threads

Use-Case # 6 User Edits the Payment Information.

1. Open DTChangePaymentInfoTestPlan.jmx.²⁶
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

²⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

The results are shown in Figure 11-16.

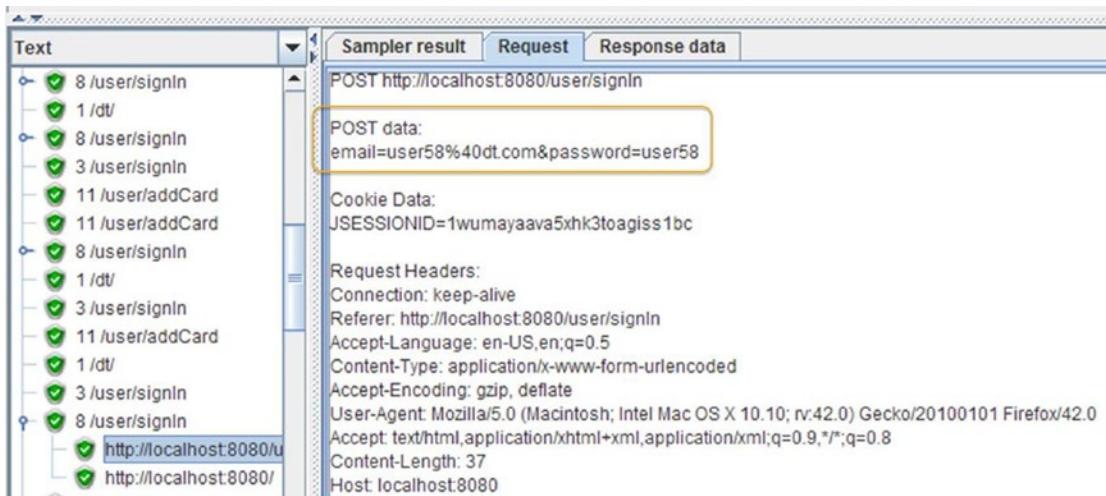


Figure 11-16. Change payment info 10 threads

Use-Case # 7 User Uses Continue Shopping Option.

1. Open DTContinueShoppingTestPlan.jmx.²⁷
2. Click on Thread Group. In Thread properties, configure **Number of Threads (users)** as 10, **Ramp-Up Period (in seconds)** as 1, and **Loop Count** as 1.
3. Save the updated test plan.
4. Run the test.

You will see the similar user requests under the View Results Tree as in the previous test results.

Implementing Actual User Behavior

Alex wanted to implement the actual user behavior, and this can be achieved by using *loop controllers* and *timers*. Loop controllers are used to loop through the product catalog page. Timers can be used to introduce a delay to simulate the time taken by the user to look for the products on the catalog page. Actual user behavior is simulated by combining loop controllers and timers.

The **Continue Shopping** use-case is a classic example of looping through the products catalog page multiple times.

Digital Toys Inc. web application does not allow adding the same product twice, so Alex had to manage the same behavior from the script. He enclosed the three requests-/dt/index, /product/detail and /order/addToCart-inside a loop controller.

²⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

Alex followed these steps to update the test.

1. Open DTContinueShoppingTestPlan.jmx²⁸ (see Figure 11-17).



Figure 11-17. Looping request

Noting that the number of products purchased depends on the user, Alex wanted to allow for this and followed these steps to achieve the actual user behavior.

1. Click on Thread Group and go to Edit ▶ Add ▶ Config Element. Add Random Variable. Configure **Variable Name** as `numberOfProduct`, **Minimum Value** as 1, **Maximum Value** as 6, and under **Per Thread (User)**? as True (see Figure 11-18).

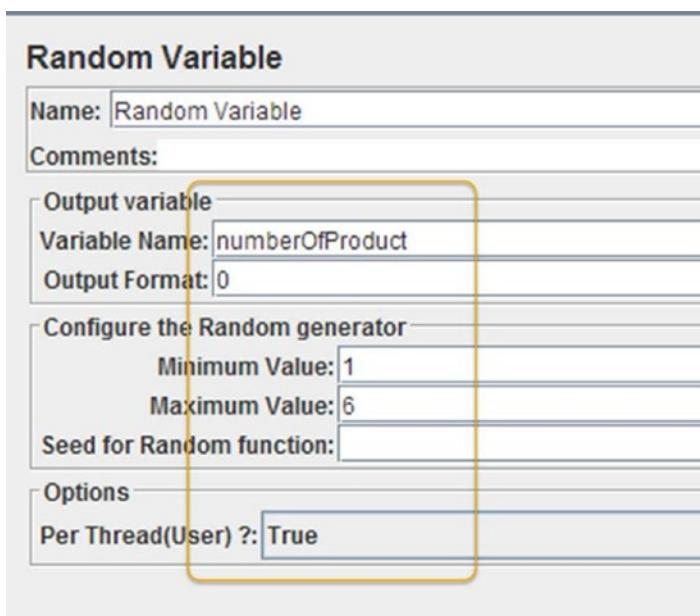


Figure 11-18. Random variable

²⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

- Click on Loop Controller. Configure **Loop Count** as the *random variable* \${numberOfProduct} (see Figure 11-19).

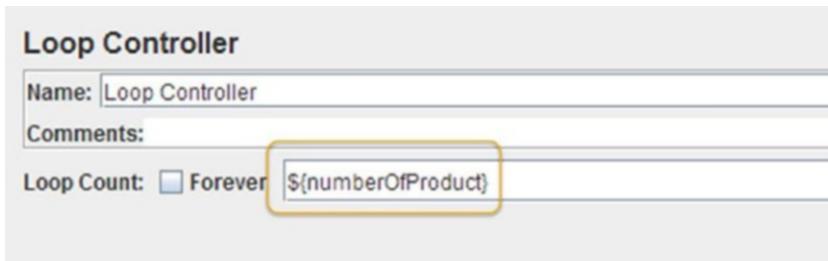


Figure 11-19. Loop count

- Click on Loop Controller and go to Edit ▶ Add ▶ Config Element. Add Counter. Configure as shown in Figure 11-20.

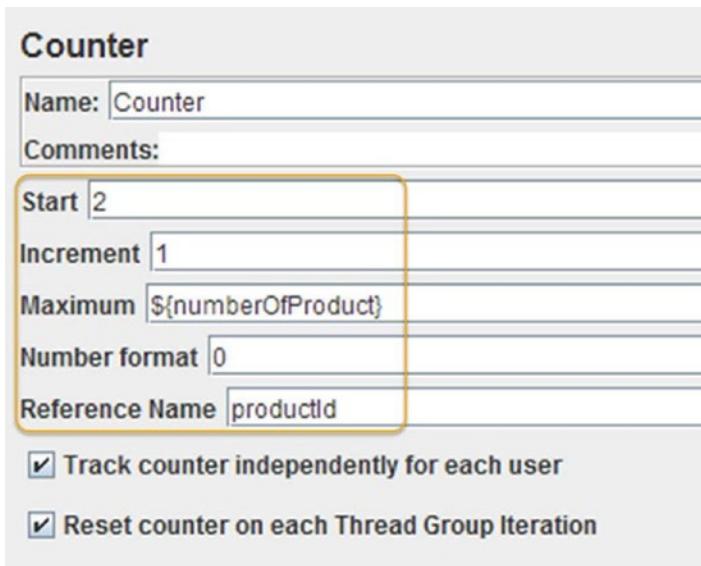
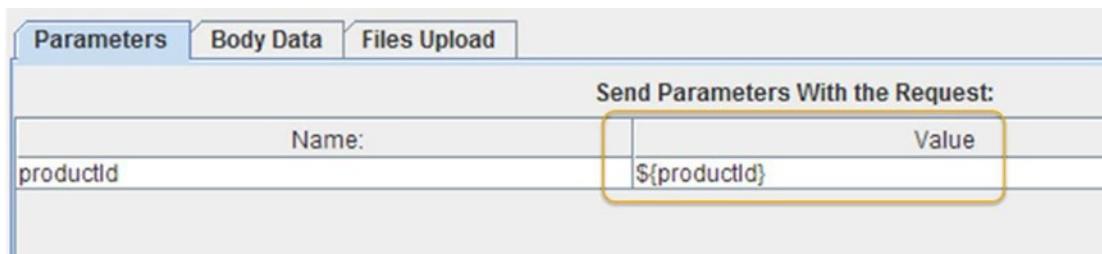


Figure 21-20. Counter

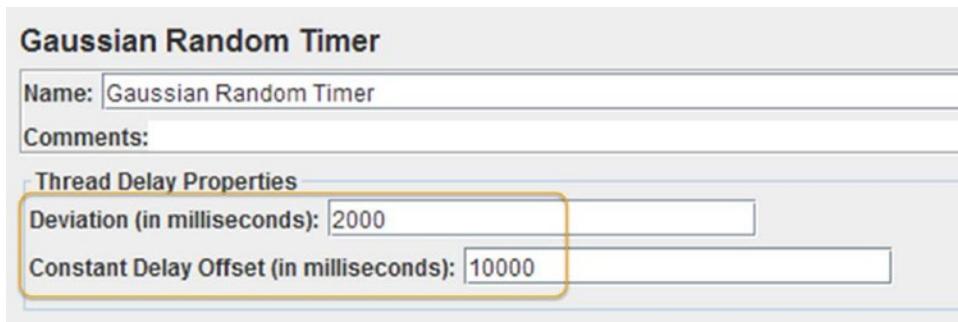
- Click on /product/detail HTTP Request. In the **Send Parameters With Request** field, click on the Add button to add a name-value pair. Configure the **Name** as productId and the **Value** as \${productId}.
- Click on /order/addToCart HTTP Request. In the **Send Parameters With Request** field, click on the Add button to add a name-value pair. Configure the **Name** as productId and the **Value** as \${productId} (see Figure 11-21).

**Figure 11-21.** HTTP request value

6. Save the updated test plan.

Users typically spend time on the product catalog page thinking about their next action. This can be simulated by using a timer. Alex decided to use a Gaussian random timer and estimated that the user would take 2 to 10 seconds. He followed these steps.

1. Click on /product/detail GET HTTP Request and go to Edit ► Add ► Timer. Add Gaussian Random Timer. Configure **Deviation (in milliseconds)** as 2000 and **Constant Delay Offset (in milliseconds)** as 10000.
2. Click on /user/addAddress POST HTTP Request and go to Edit ► Add ► Timer. Add Gaussian Random Timer. Configure **Deviation (in milliseconds)** as 2000 and **Constant Delay Offset (in milliseconds)** as 10000.
3. Click on /user/addCard POST HTTP Request and go to Edit ► Add ► Timer. Add Gaussian Random Timer. Configure **Deviation (in milliseconds)** as 2000 and **Constant Delay Offset (in milliseconds)** as 10000.
4. Click on /order/orderHistory GET HTTP Request and go to Edit ► Add ► Timer and add Gaussian Random Timer. Configure **Deviation (in milliseconds)** as 2000 and **Constant Delay Offset (in milliseconds)** as 10000 (see Figure 11-22).

**Figure 11-22.** Gaussian random timer

Similarly, Alex updated remaining test plans as per Table 11-10.

Table 11-10. Test Plan Vs. Request Mapping

Test Plan	POST Request Sampler(s)
DTFirstOrderTestPlan.jmx	/product/detail, /user/addAddress, /user/addCard, /order/orderHistory
DTSecondTimeOrderTestPlan.jmx	/product/detail, /order/orderHistory
DTChangeAddressTestPlan.jmx	/user/addAddress
DTChangePaymentInfoTestPlan.jmx	/user/addCard

Alex added a Gaussian random timer on all the pages that required user thinktime in all the test plans.

After updating the JMeter test script, Alex was confident that the script would implement the actual user behavior. For each of the tests, he prepared a list of users and updated the *CSV Data Set Config* configuration as per Table 11-11.

Table 11-11. Test Script vs. Input CSV File Mapping

JMeter Test Script Name	Input CSV File Name
DTFirstOrderTestPlan.jmx	first-order.csv
DTSecondTimeOrderTestPlan.jmx	secondtime-order.csv
DTChangeAddressTestPlan.jmx	change-address.csv
DTChangePaymentInfoTestPlan.jmx	change-paymentinfo.csv
DTContinueShoppingTestPlan.jmx	continue-shopping.csv

It makes sense to run the test for Placing the Order the Second Time, Editing the Address Information, and Editing the Payment Information only after running the test for Placing the Order the First Time.

Tip Ensure that the sequence in which the tests are run follows the real user scenario.

Results Metrics

Once the tests are agreed upon, they are standardized and the results noted as a baseline. The test results, together with the required performance metrics, are shared with the engineering team. When the engineering team implements a fix, the tests are re-run and the results are compared with the baseline.

Alex had a discussion with the team and standardized on the performance criteria to be measured. These were the Average, Min, and Max response times of each HTTP request. They also needed the utilization metrics for CPU, memory, and network.

Alex added an *aggregate report*, which gave him Average, Min, and Max response times of each HTTP request. He added *jp@gc - PerfMon Metrics Collector* to get CPU, memory, and network utilization stats.

Alex disabled the View Results Tree report for the production run, as it is memory intensive.

As JMeter does not ship with PerMon Agent out of the box, Alex downloaded the server agent from Server Agent URL.²⁹

In order to add *jp@gc - PerfMon Metrics Collector* to JMeter, Alex downloaded the plugin from JMeterPlugins URL³⁰ and extracted it to the JMeter main directory.

Alex followed these steps to update the existing test plans.

Use-Case # 1 User Lands on the Home Page and Browses Products.

1. Open DTHomePageTestPlan.jmx.³¹
2. Click on View Results Tree and go to Edit ► Disable.
3. Click on Thread Group and go to Edit ► Add ► Listener. Add Aggregate Report.
4. Click on Thread Group and go to Edit ► Add ► Listener. Add *jp@gc - PerfMon Metrics Collector*.
5. Click the Add Row button on the respective servers to monitor for CPU, memory, and network.
6. Run the test.

The results are shown in Figure 11-23.

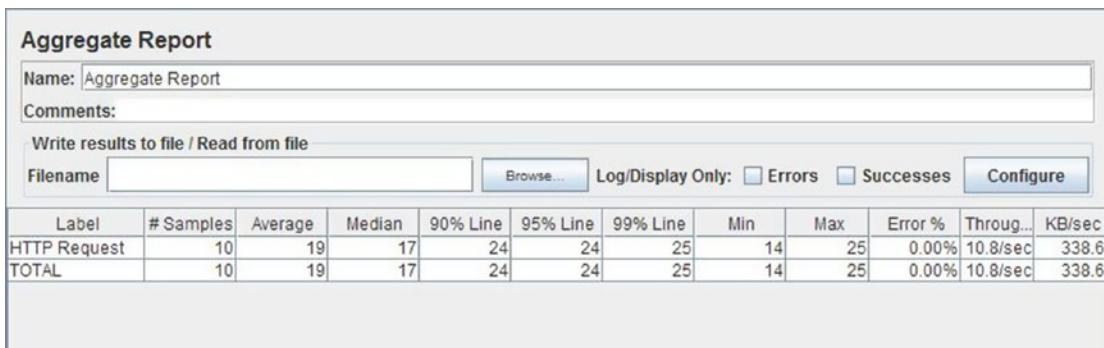


Figure 11-23. Home page aggregate report

The PerfMon Metrics Collectors report does not show many of the changes if you just trigger with 10 threads. Try to increase the threads to 100 and then run the test; you will find the results shown in Figure 11-24.

²⁹<http://jmeter-plugins.org/downloads/file/ServerAgent-2.2.1.zip>

³⁰<http://jmeter-plugins.org/downloads/file/JMeterPlugins-Standard-1.3.1.zip>

³¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTHomePageTestPlan.jmx

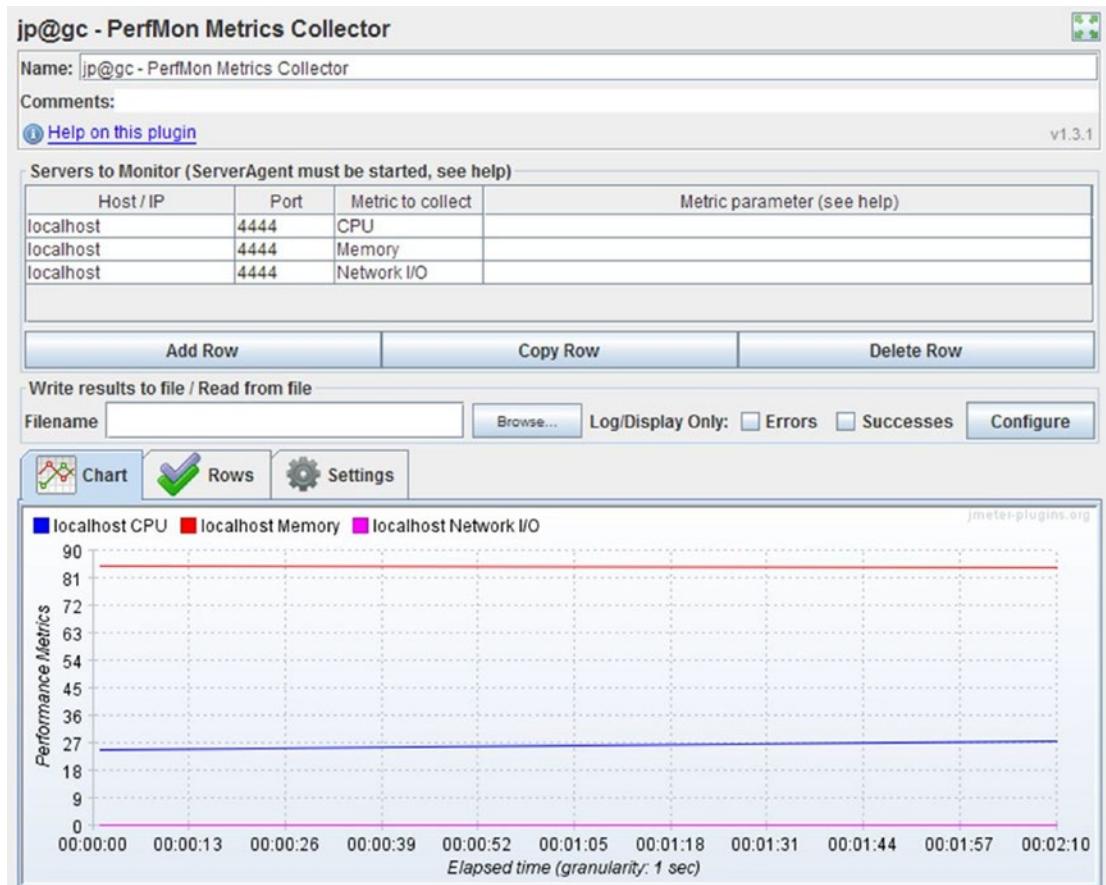


Figure 11-24. Home page PerfMon report (see color image in source code file)

In the same way, he updated the remaining test plans and generated the reports.
Use-Case # 3 User Places an Order for the First Time.

1. Open DTFirstOrderTestPlan.jmx.³²
2. Click on View Results Tree and go to Edit ▶ Disable.
3. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add jp@gc - PerfMon Metrics Collector.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add Aggregate Report.
5. Click on the Add Row button on the respective servers to monitor for CPU, memory, and network.
6. Save the updated test plan.
7. Run the test.

³²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

The results are shown in Figures 11-25 and 11-26.

Aggregate Report											
Name: Aggregate Report											
Comments:											
Write results to file / Read from file											
Filename <input type="text"/>				Browse...	Log/Display Only: <input type="checkbox"/> Errors <input type="checkbox"/> Successes			Configure			
Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
1 /dt	10	19	17	20	20	35	16	35	0.00%	10.8/sec	335.7
3 /user/signIn	10	6	6	7	7	8	5	8	0.00%	11.1/sec	68.0
8 /user/signIn	10	29	29	33	33	35	25	35	0.00%	10.8/sec	345.5
11 /product/detail	10	17	16	22	22	23	14	23	0.00%	1.6/sec	10.0
13 /order/addToCart	10	22	22	25	25	34	17	34	0.00%	1.6/sec	10.5
15 /order/checkOut	10	26	23	29	29	51	20	51	0.00%	1.6/sec	10.9
21 /user/addAddress	10	50	38	86	86	107	33	107	0.00%	2.2/sec	15.9
25 /user/addCard	10	44	41	49	49	80	32	80	0.00%	1.6/sec	11.5
28 /order/placeOrder	10	82	67	126	126	242	38	242	0.00%	1.6/sec	52.0
31 /order/orderHistory	10	65	44	157	157	159	36	159	0.00%	54.0/m...	9.0
33 /user/signOut	10	29	20	40	40	98	17	98	0.00%	54.0/m...	28.2
TOTAL	110	35	27	67	86	159	5	242	0.00%	2.2/sec	35.8

Figure 11-25. First order aggregate report

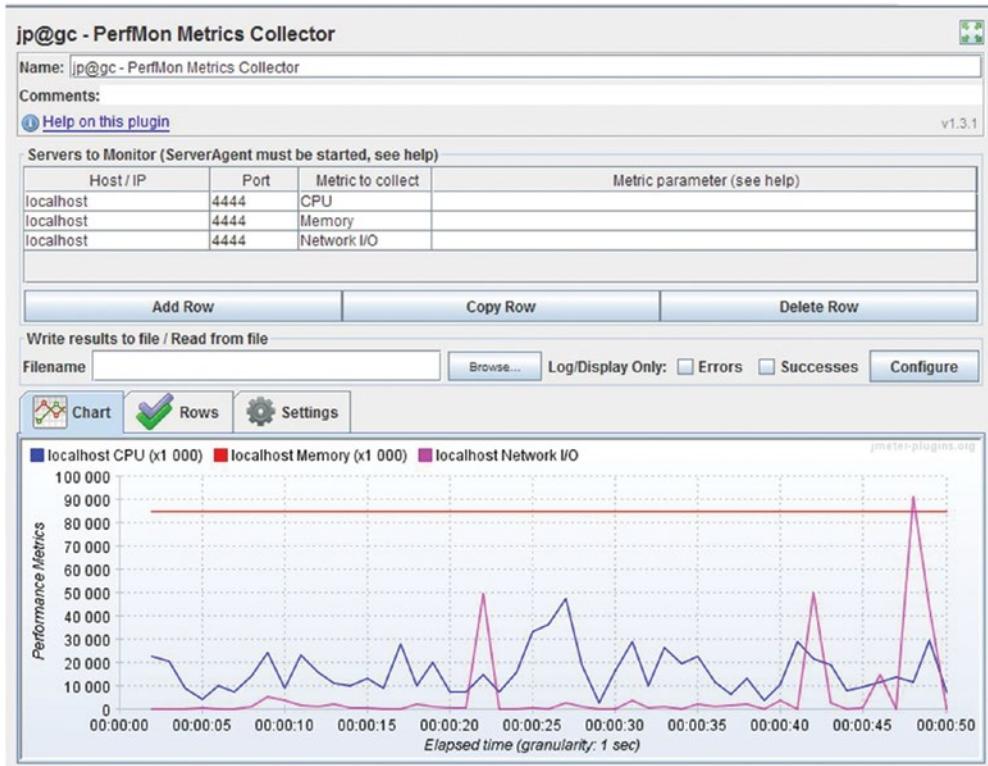


Figure 11-26. First order PerfMon report (see color image in source code file)

Use-Case # 4 User Places an Order for the Second Time.

1. Open DTSecondTimeOrderTestPlan.jmx.³³
2. Click on View Results Tree and go to Edit ► Disable.
3. Click on Thread Group and go to Edit ► Add ► Listener. Add jp@gc - PerfMon Metrics Collector.
4. Click on Thread Group and go to Edit ► Add ► Listener. Add Aggregate Report.
5. Click on the Add Row button on the respective servers to monitor for CPU, memory, and network.
6. Save the updated test plan.
7. Run the test.

The results are shown in Figures 11-27 and 11-28.

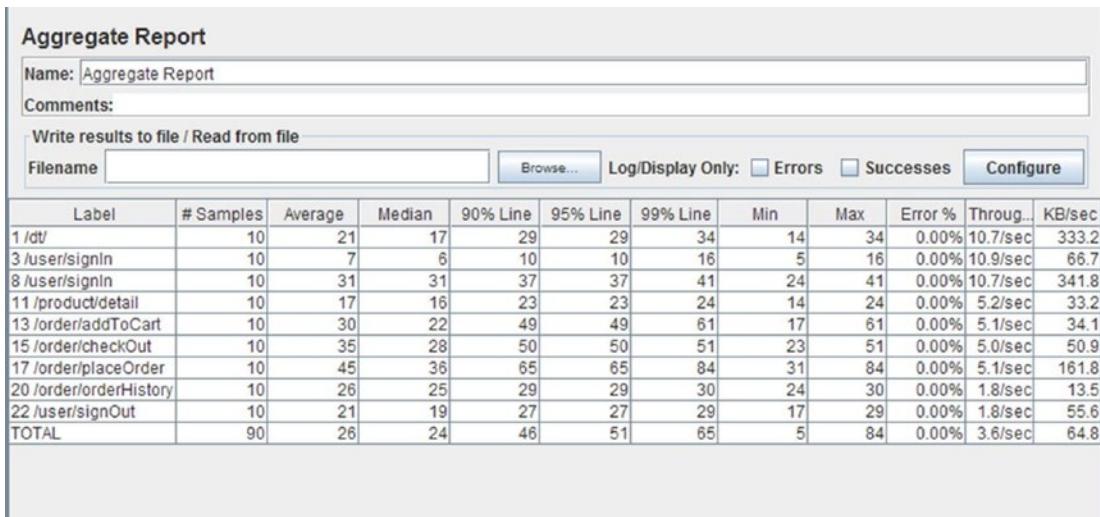


Figure 11-27. Second time order aggregate report

³³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTSecondTimeOrderTestPlan.jmx

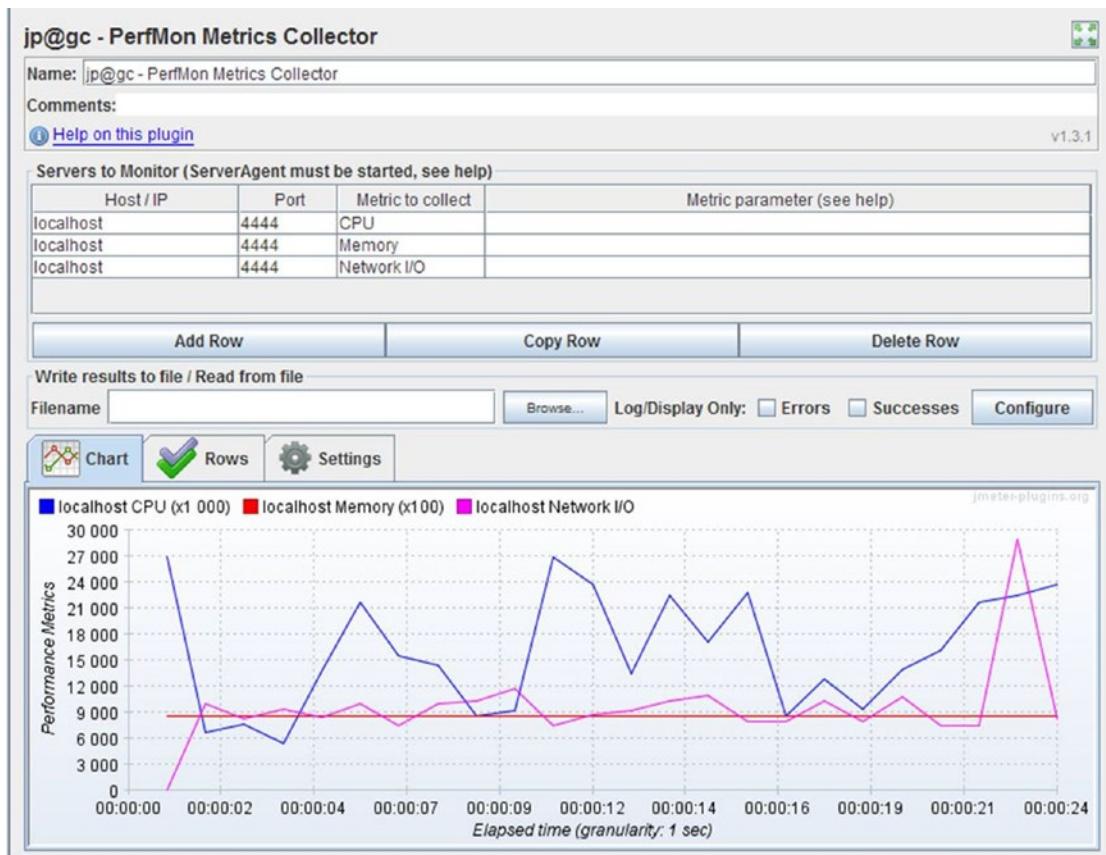


Figure 11-28. Second time order PerfMon report (see color image in source code file)

Use-Case # 5 User Edits the Billing/Shipping Address.

1. Open DTChangeAddressTestPlan.jmx.³⁴
2. Click on View Results Tree and go to Edit > Disable.
3. Click on Thread Group and go to Edit > Add > Listener. Add Aggregate Report.
4. Click on Thread Group and go to Edit > Add > Listener. Add jp@gc - PerfMon Metrics Collector.
5. Click the Add Row button on jp@gc - PerfMon Metrics Collector and add the respective servers to monitor for CPU, memory, and network.
6. Save the updated test plan.
7. Run the test.

³⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

The results are shown in Figures 11-29 and 11-30.

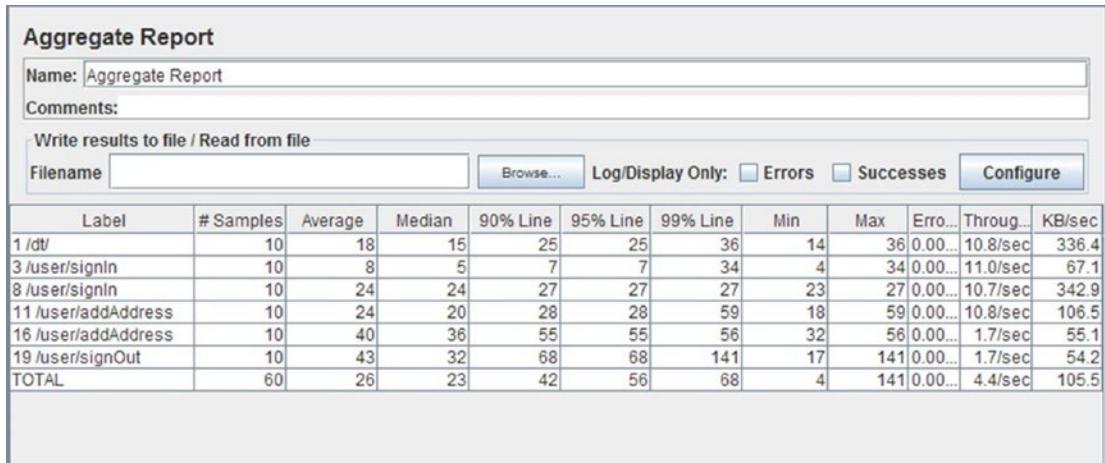


Figure 11-29. Change address aggregate report

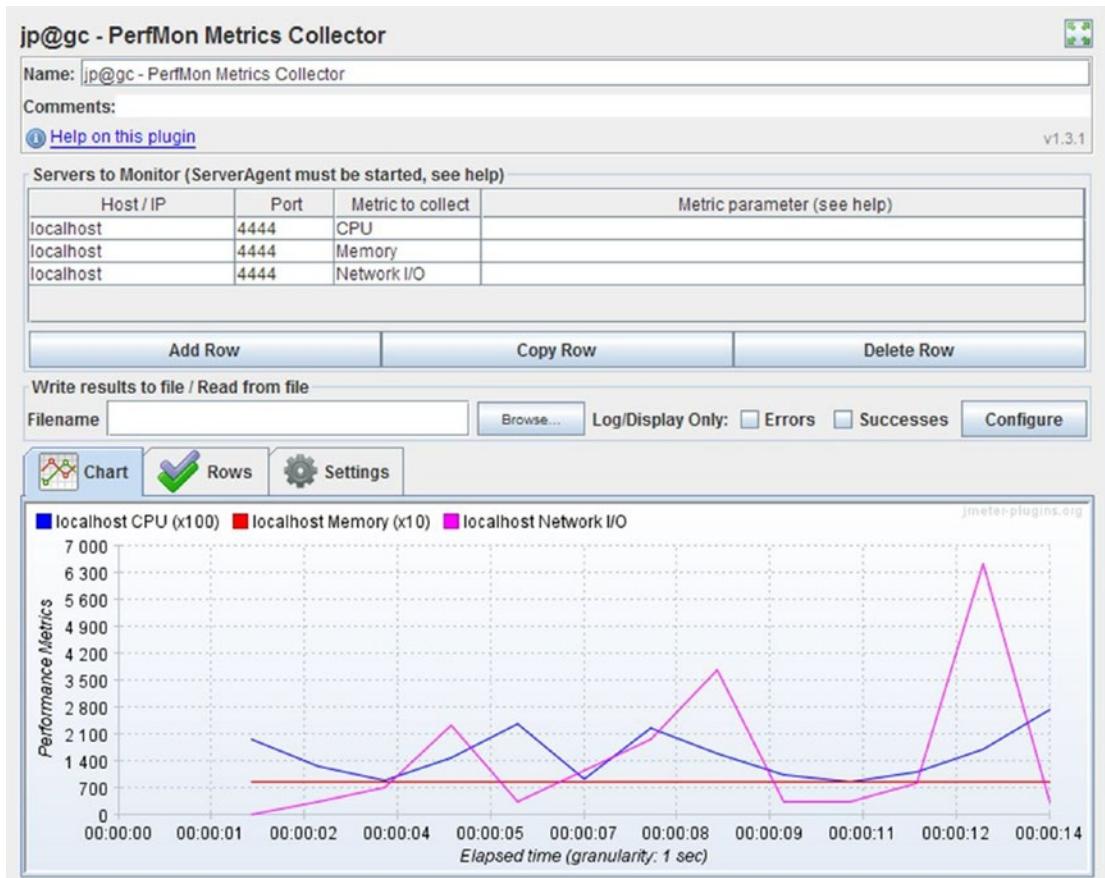


Figure 11-30. Change address PerfMon report (see color image in source code file)

Use-Case # 6 User Edits the Payment Information.

1. Open DTChangePaymentInfoTestPlan.jmx.³⁵
2. Click on View Results Tree and go to Edit ► Disable.
3. Click on Thread Group and go to Edit ► Add ► Listener. Add Aggregate Report.
4. Click on Thread Group and go to Edit ► Add ► Listener. Add jp@gc - PerfMon Metrics Collector.
5. Click the Add Row button on *jp@gc - PerfMon Metrics Collector* and add the respective servers to monitor for CPU, memory, and network.
6. Save the updated test plan.
7. Run the test.

The results are shown in Figures 11-31 and 11-32.

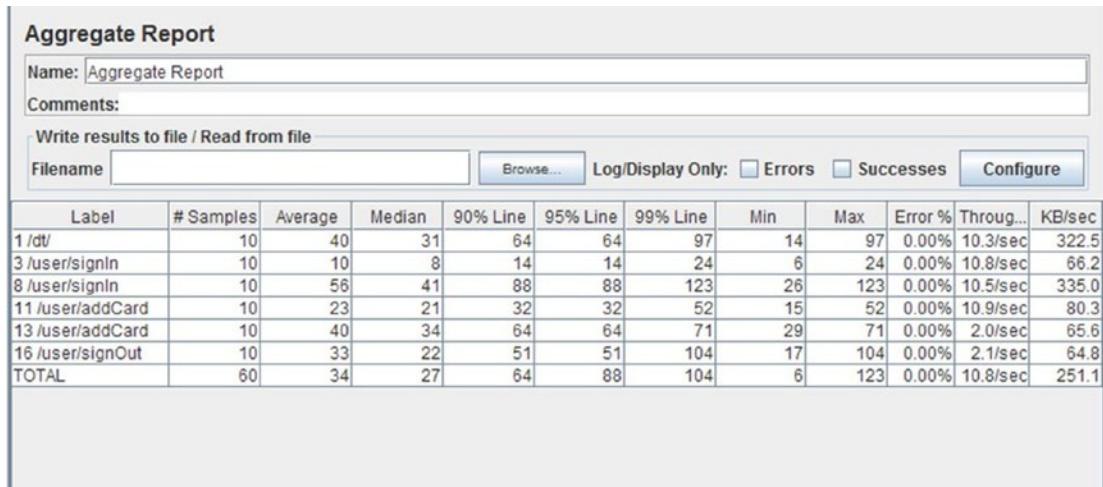


Figure 11-31. Change payment info aggregate report

³⁵https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

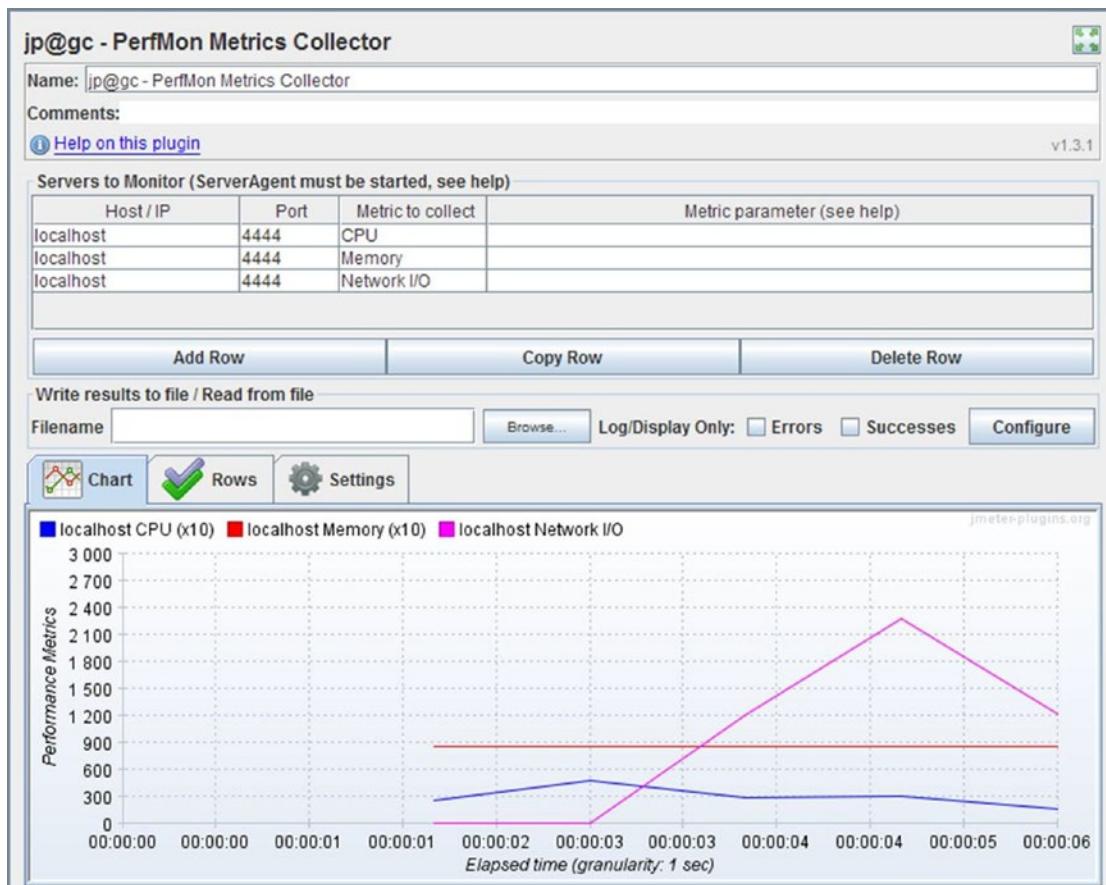


Figure 11-32. Change payment info PerfMon report (see color image in source code file)

Use-Case # 7 User Uses the Continue Shopping Option.

1. Open DTContinueShoppingTestPlan.jmx.³⁶
2. Click on View Results Tree and go to Edit ► Disable

³⁶https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

3. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add Aggregate Report.
4. Click on Thread Group and go to Edit ▶ Add ▶ Listener. Add jp@gc - PerfMon Metrics Collector.
5. Click on the Add Row button on *jp@gc - PerfMon Metrics Collector* and add the respective servers to monitor for CPU, memory, and network.
6. Save the updated test plan.
7. Run the test.

Results are shown in Figures 11-33 and 11-34.

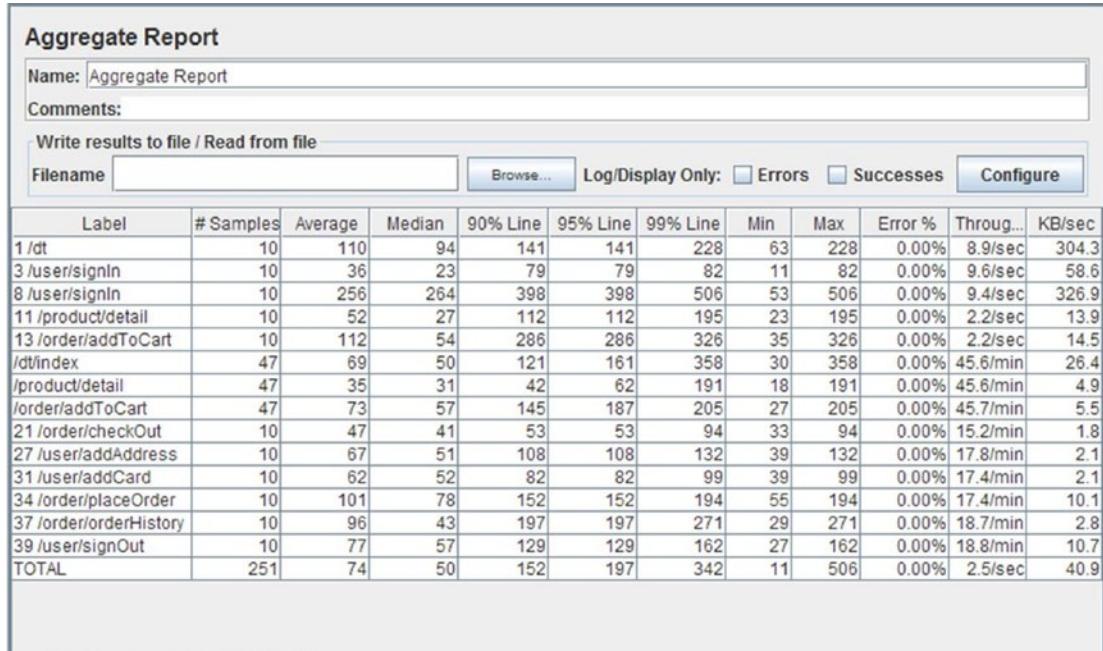


Figure 11-33. Continue shopping aggregate report

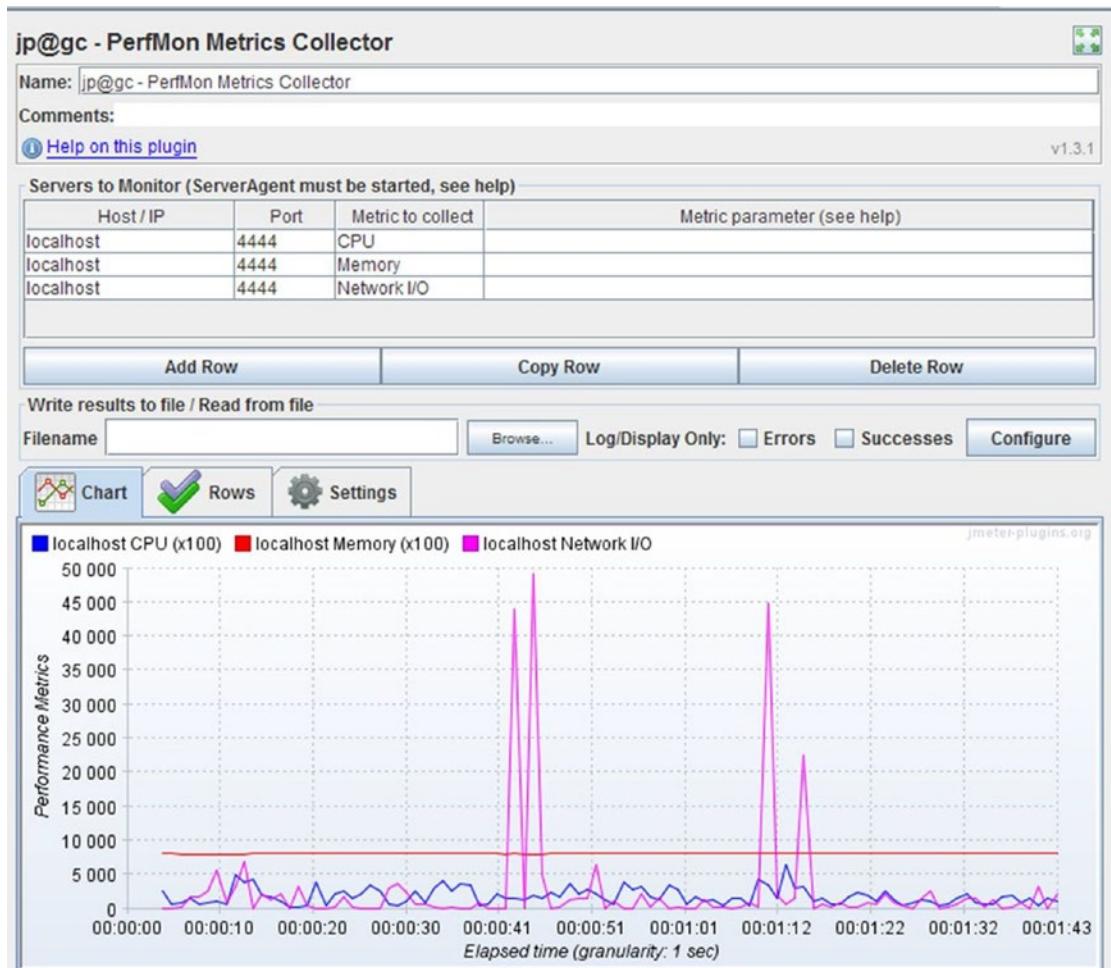


Figure 11-34. Continue shopping PerfMon report (see color image in source code file)

Organizing Tests

Alex captured the results from the aggregate report and the *jp@gc - PerfMon Metrics Collector* report and showed them to his manager. Bob wanted Alex to identify the problematic use-cases, web pages, and their associated HTTP requests for further analysis by the team.

Each use-case is a sequence of steps and each step comprises multiple HTTP requests. Alex prepared a table showing the steps and the HTTP requests associated with each step.

Alex combined HTTP requests associated with a use-case under a transaction controller and named them with the step name. This allows the report to display results on a per-step basis. Alex followed the table and modified the other test scripts in a similar way.

Use-Case # 1 User Lands on the Home Page and Browses the Products.

Table 11-12. Use-Case # 1 Test Plan Steps Vs. HTTP Request Mapping

No.	Step Name	HTTP Request
1	Home Page	/dt

1. Open DTHomePageTestPlan.jmx.³⁷
2. Click on Thread Group and go to Edit ► Add ► Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** as true and **Name** as Home Page.
3. Save the updated test plan.
4. Run the test.

Results are shown in Figure 11-35.

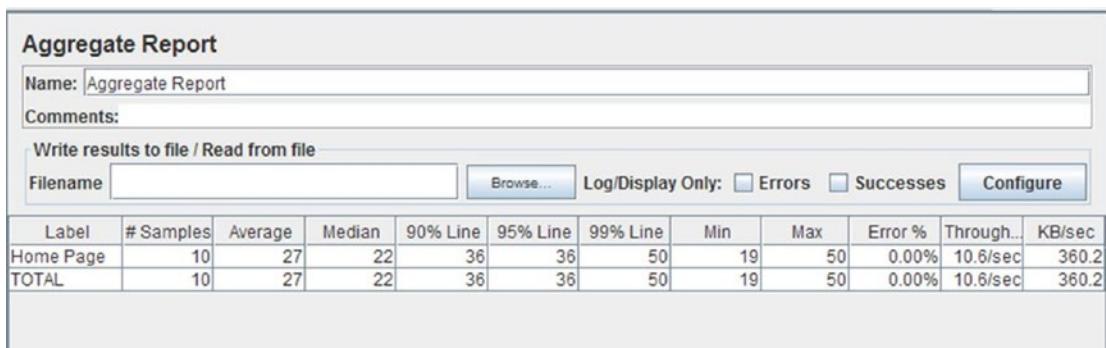


Figure 11-35. Home page aggregate report

³⁷https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTHomePageTestPlan.jmx

Use-Case # 3 User Places an Order for the First Time.

Table 11-13 shows the steps and the associated HTTP requests.

Table 11-13. Use-Case # 2 Test Plan Steps Vs. HTTP Request Mapping

No.	Manual Step	HTTP Request
1	Home Page	/dt
2	Login Page	/user/signIn, /user/signIn
3	Index/Product Catalog Page	/dt/index
4	Details	/product/detail
5	Add To Cart	/order/addToCart
6	Check Out	/order/checkOut
7	Billing/Shipping Address	/user/addAddress, /user/addAddress, /order/checkOut
8	Credit Card	/user/addCard, /user/addCard, /order/ checkOut
9	Place Order	/order/placeOrder, /dt/index
10	Order History	/order/orderHistory
11	Sign Out	/user/signOut, /dt/index

Alex added a transaction controller for each of the steps.

1. Open DTFirstOrderTestPlan.jmx.³⁸
2. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** to make it true.
3. Click on Transaction Controller and go to Edit ▶ Copy. Follow this step 10 times.
4. Follow Table 11-13 and update the name of each transaction controller and the associated request.
5. Save the updated test plan.
6. Run the test.

³⁸https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTFirstOrderTestPlan.jmx

The results are shown in Figure 11-36.

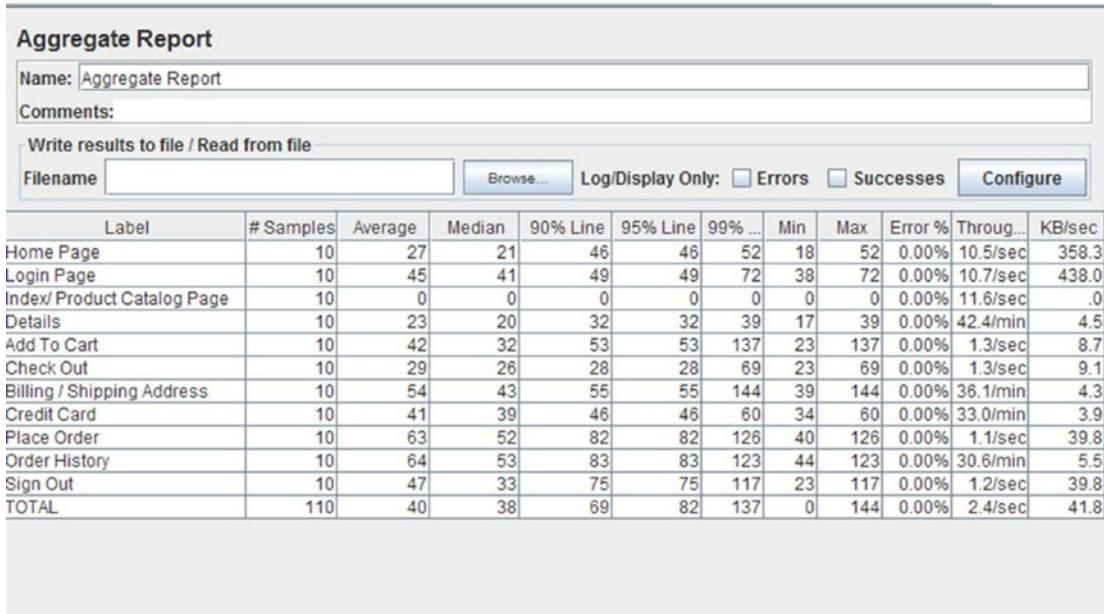


Figure 11-36. First order useful report

Use-Case # 4 User Places an Order for the Second Time.

Table 11-14 shows the steps and the associated HTTP requests.

Table 11-14. Use-Case # 4 Test Plan Steps Vs. HTTP Request Mapping

No.	Step Name	HTTP Request
1	Home Page	/dt
2	Login Page	/user/signIn, /user/signIn
3	Index/Product Catalog Page	/dt/index
4	Details	/product/detail
5	Add To Cart	/order/addToCart
6	Check Out	/order/checkOut
7	Place Order	/order/placeOrder, /dt/index
8	Order History	/order/orderHistory
9	Sign Out	/user/signOut, /dt/index

1. Open DTSecondTimeOrderTestPlan.jmx.³⁹
2. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** to make it true.
3. Click on Transaction Controller and go to Edit ▶ Copy. Follow this step 8 times.
4. Follow Table 11-14 and update the name of each transaction controller and the associated request.
5. Before running the test, Alex ran the First Order test with the same set of users assigned for this test script.
6. Save the updated test plan.
7. Run the test.

The results are shown in Figure 11-37.

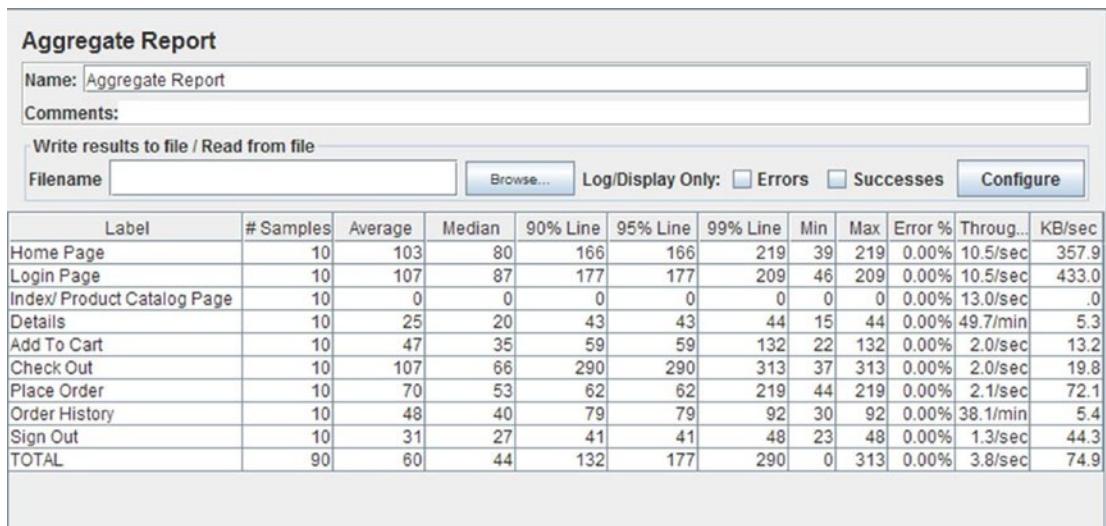


Figure 11-37. Second time order useful report

³⁹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTSecondTimeOrderTestPlan.jmx

Use-Case # 5 User Edits the Billing/Shipping Address.

Table 11-15 shows steps and the associated HTTP requests.

Table 11-15. Use-Case # 5 Test Plan Steps vs. HTTP Request Mapping

No.	Step Name	HTTP Request
1	Home Page	/dt
2	Login Page	/user/signIn, /user/signIn
3	Index/Product Catalog Page	/dt/index
4	Billing / Shipping Address	/user/addAddress, /user/addAddress, /dt/ index
5	Sign Out	/user/signOut, /dt/index

1. Open DTChangeAddressTestPlan.jmx.⁴⁰
2. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** to make it true.
3. Click on Transaction Controller and go to Edit ▶ Copy. Follow this step four times.
4. Follow Table 11-15 and update the name of each transaction controller and the associated request.
5. Before running the test, Alex ran the First Order test with the same set of users assigned for this test script.
6. Save the updated test plan.
7. Run the test.

The results are shown in Figure 11-38.

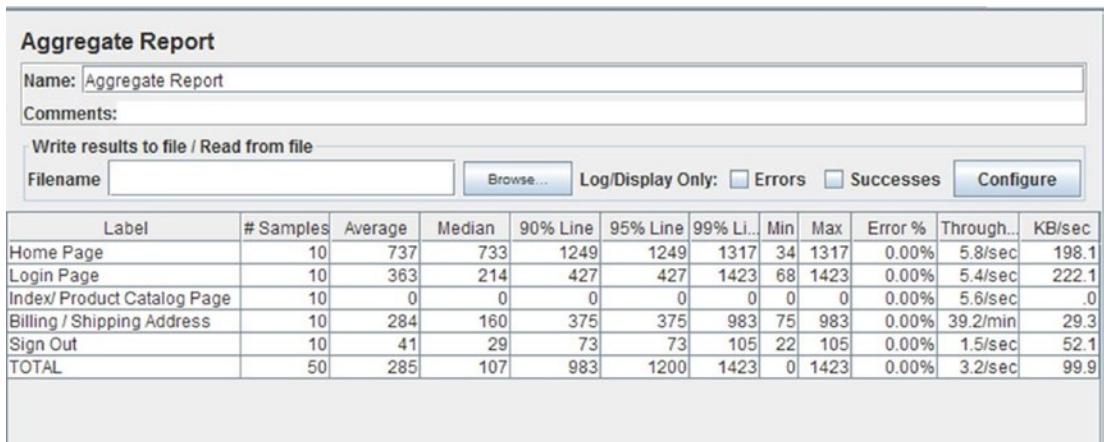


Figure 11-38. Change address useful report

⁴⁰https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangeAddressTestPlan.jmx

Use-Case # 6 User Edits Payment Information.

Table 11-16 shows the steps and the associated HTTP requests.

Table 11-16. Use-Case # 6 Test Plan Steps Vs. HTTP Request Mapping

No.	Manual Step	HTTP Request
1	Home Page	/dt
2	Login Page	/user/signIn, /user/signIn
3	Index/Product Catalog Page	/dt/index
4	Add Credit Card	/user/addCard, /user/addCard, /dt/index
5	Sign Out	/user/signOut, /dt/index

1. Open DTChangePaymentInfoTestPlan.jmx.⁴¹
2. Click on Thread Group and go to Edit ► Add ► Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** to make it true.
3. Click on Transaction Controller and go to Edit ► Copy. Follow this step four times.
4. Follow Table 11-16 and update the name of each transaction controller and the associated request.
5. Before running the test, Alex ran the First Order test with the same set of users assigned for this test script.
6. Save the updated test plan.
7. Run the test.

The results are shown in Figure 11-39.

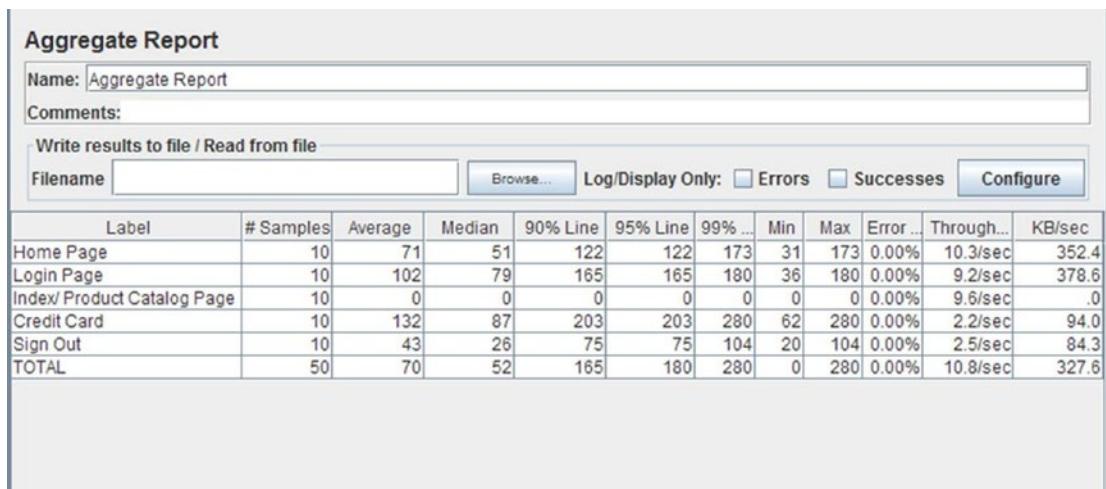


Figure 11-39. Change payment info aggregate report

⁴¹https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTChangePaymentInfoTestPlan.jmx

Use-Case # 7 User Uses the Continue Shopping Option.

Table 11-17 shows the steps and the associated HTTP requests.

Table 11-17. Use-Case # 7 Test Plan Steps Vs. HTTP Request Mapping

No.	Manual Step	HTTP Request
1	Home Page	/dt
2	Login Page	/user/signIn, /user/signIn
3	Index/Product Catalog Page	/dt/index
4	Details	/product/detail
5	Add To Cart	/order/addToCart
6	Continue Shopping	/dt/index, /product/detail, /order/ addToCart
7	Check Out	/order/checkOut
8	Billing/Shipping Address	/user/addAddress, /user/addAddress, / order/checkOut
9	Credit Card	/user/addCard, /user/addCard, /order/ checkOut
10	Place Order	/order/placeOrder, /dt/index
11	Order History	/order/orderHistory
12	Sign Out	/user/signOut, /dt/index

1. Open DTContinueShoppingTestPlan.jmx.⁴²
2. Click on Thread Group and go to Edit ▶ Add ▶ Logic Controller. Add Transaction Controller. Configure **Generate Parent Sample** to make it true.
3. Click on Transaction Controller and go to Edit ▶ Copy. Follow this step 11 times.
4. Follow Table 11-17 and update the name of each transaction controller and the associated request.
5. Before running the test, Alex ensured that he ran the use-case User Placing the First Order test with the same set of users as used in this test script.
6. Save the updated test plan.
7. Run the test.

⁴²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

The results are shown in Figure 11-40.

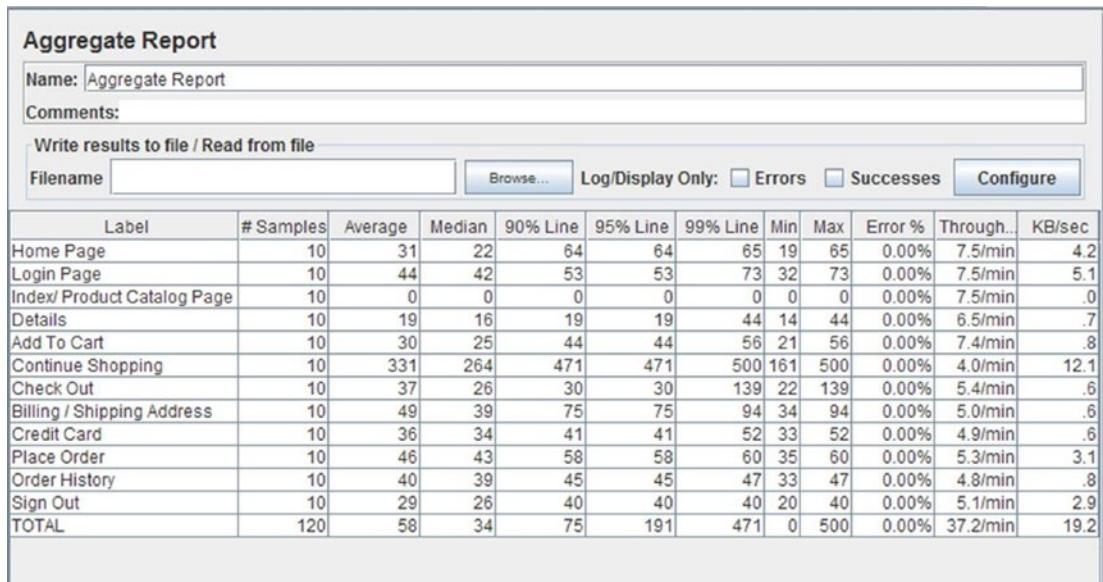


Figure 11-40. Continue shopping aggregate report

Before running the updated JMeter test scripts, Alex initialized his test environment by deploying a clean schema. He wanted to be doubly sure that the test environment and the data were the same between the test runs.

Alex ran the tests with 20 users to warm up the test environment. After that, he ran all the scripts with 500 users and approached Bob. Bob was happy to see results but asked for another set of tests where the users were doing different operations at the same time.

Tip Run the test scripts with a few users as a warmup before running the real performance test.

Combining Multiple Tests

Alex simulated the load pattern generated by real users in production by running multiple thread groups in parallel, with each of the thread group executing a different use-case. The percentage of use-case executions was based on the user distribution table that he prepared earlier.

Alex followed these steps to merge all his test plans into a single test plan.

1. Create a test plan and give it a meaningful name, such as Digital Toys Main.
2. Click on Test Plan and go to Edit ▶ Merge. Select a test plan, as shown in Table 11-18.
3. Save the test plan as DTMainTestPlan.jmx.⁴³

⁴³https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTMainTestPlan.jmx

He added each test plan listed in Table 11-18 as a thread group under Digital Toys Main.

Table 11-18. Test Plan Steps Vs. Thread Group Mapping

Test Plan	Thread Group
DTHomePageTestPlan.jmx	Home Page
DTFirstOrderTestPlan.jmx	First Order
DTSecondTimeOrderTestPlan.jmx	Second Time Order
DTChangeAddressTestPlan.jmx	Aggregate Report
DTChangePaymentInfoTestPlan.jmx	Change Payment Info
DTContinueShoppingTestPlan.jmx	Continue Shopping

After making the required changes, the test plan looked as shown in Figure 11-41.

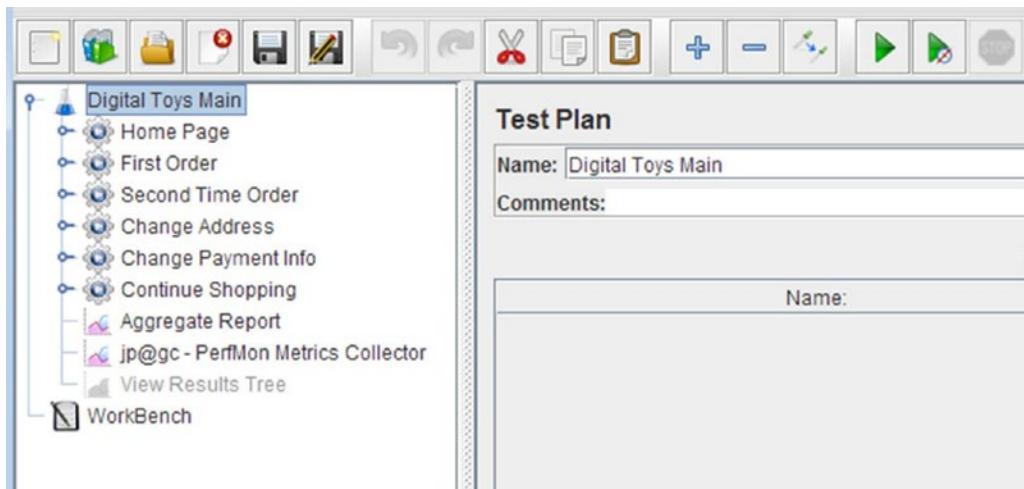


Figure 11-41. Multiple thread group test plan

Before running the updated JMeter test scripts, Alex initialized his test environment by deploying clean schema.

The use-cases for DTSecondTimeOrderTestPlan.jmx, DTChangeAddressTestPlan.jmx, and DTChangePaymentInfoTestPlan.jmx require that an initial order be placed in the system. For this, Alex runs the DTFirstOrderTestPlan.jmx test with an input user file. This also serves to warm up the test environment.

In order to run the test in parallel mode, Alex ensured that the Run Thread Groups Consecutively (i.e. run groups one at a time) option was unchecked (which is the default) (see Figure 11-42).

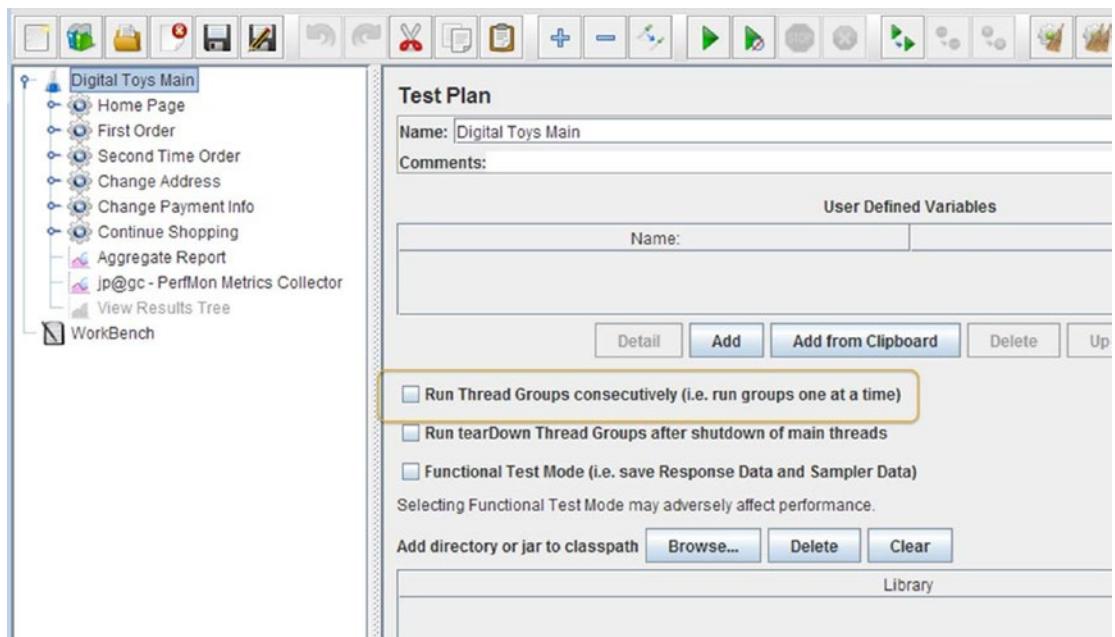


Figure 11-42. Thread groups parallel

He then ran the test with varying number of threads (users). This ensures that the thread groups are picked up by JMeter in a random fashion. Before executing the tests, Alex assumed that he was testing with 500 users, so he updated the Number of Threads (Users) as per User Load Pattern table. See Table 11-19.

Table 11-19. Performance Load Distribution Chart

Browsing catalog	10%
Ordering first time	60%
Ordering second time	10%
Changing address	5%
Changing payment information	5%
Continue shopping	10%

He added an aggregate report and jp@gc - PerfMon Metrics Collector to generate the reports.

1. Click on Test Plan and go to Add ► Listener. Add Aggregate Report.
2. Click on Test Plan and go to Add ► Listener. Add jp@gc - PerfMon Metrics Collector.
3. Disable Aggregate Report and jp@gc - PerfMon Metrics Collector for the individual thread group to save on CPU/memory consumption.
4. Save the updated test plan.
5. Run the test.

The results are shown in Figures 11-43 and 11-44.

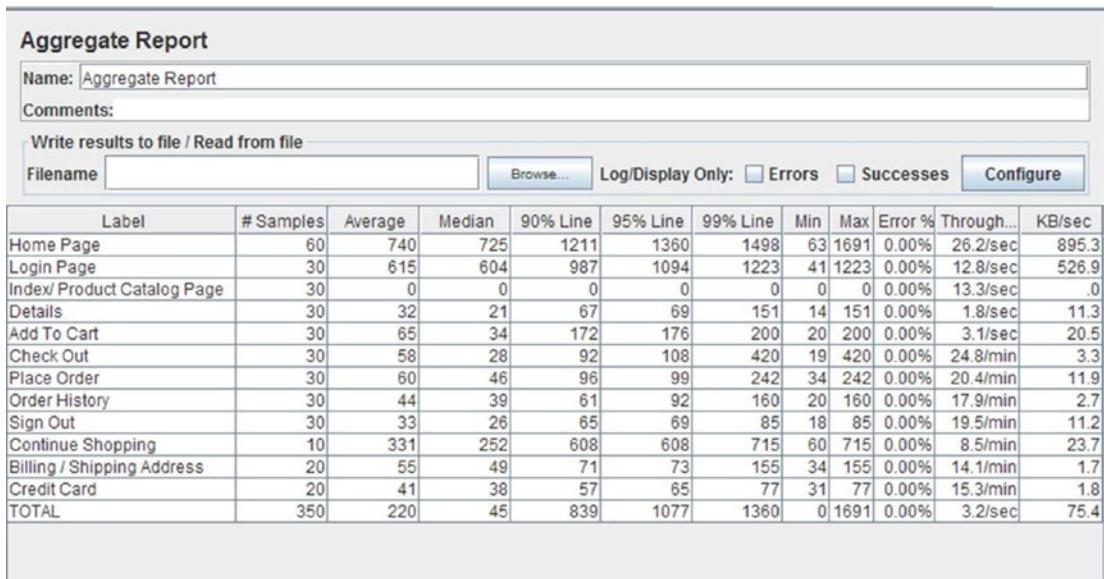


Figure 11-43. Digital Toys main test

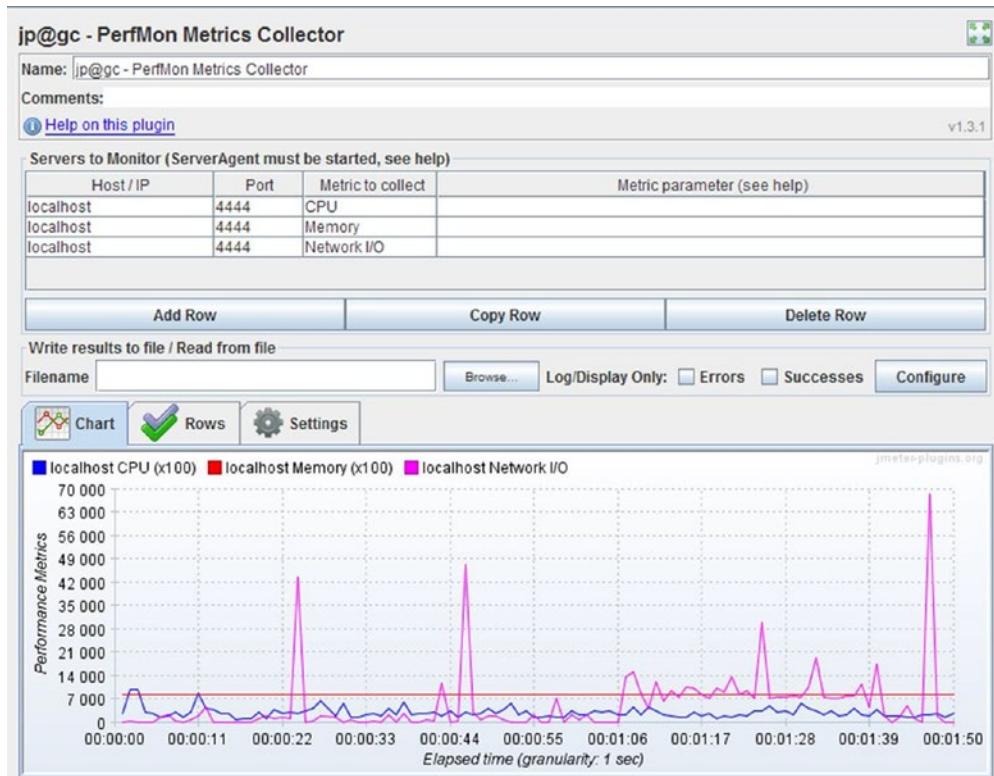


Figure 11-44. Digital Toys main test (see color image in source code file)

He collected performance results and showed them to Bob.

Questions

Bob and the team asked Alex a few questions:

Question: How do you obtain a count of logged-in users at any point in time?

Answer: Alex replied that this could be obtained from the *jp@gc - Active Threads Over Time* report.

Question: What is the count of users doing various activities (browsing the catalog, ordering, changing address, entering payment information, etc.) on the web site?

Answer: Alex said that this could be obtained by using the User Load Pattern table.

To simulate a load of 100,000 users, the number of threads (users) for each use-case can be calculated using the % distribution detailed in the table.

Question: How do you simulate varying user-load patterns based on the time of the day and duration?

Answer: Alex replied that he would configure the Scheduler field of Thread Group.

Question: How do you run the performance test for multiple days?

Answer: Alex replied that he could configure the *thread group scheduler*, *loop controller*, and *timer* to continuously run the test for a few days.

Note For a set of standard graphs and performance metrics, refer to Chapter 12, “Performance Dashboard”.

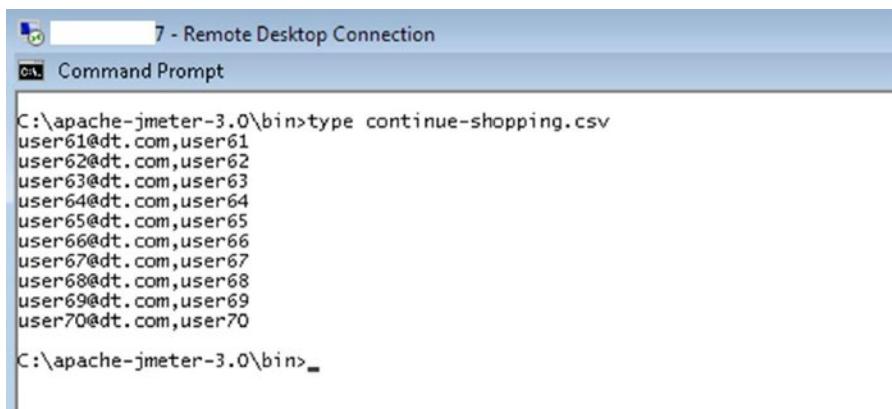
Using Distributed Environment

Alex had now established a test framework and a test suite, which he could run time and again for each new build.

Alex updated the performance testing environment from one machine to multiple machines to generate more load on the application. For this, he configured JMeter in *master-slave* mode.

DTContinueShoppingTestPlan.jmx⁴⁴ is a good candidate for starting with performance testing in distributed testing environment.

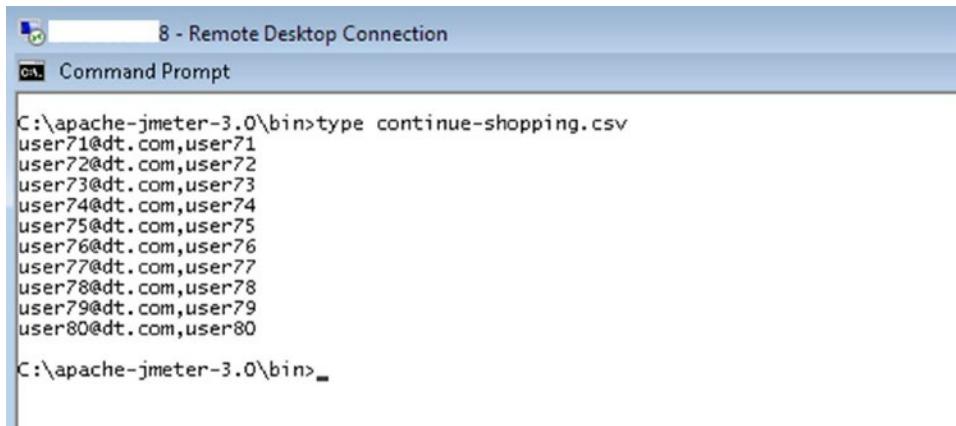
Before starting, Alex created a distinct user list for each of the slaves. He started with vm01 (Figure 11-45) and vm02 (Figure 11-46) as two slaves and placed continue-shopping.csv in the \$JMETER_HOME/bin directory of each of these VMs. This helps in isolating users from each of the slaves.



```
C:\apache-jmeter-3.0\bin>type continue-shopping.csv
user61@dt.com,user61
user62@dt.com,user62
user63@dt.com,user63
user64@dt.com,user64
user65@dt.com,user65
user66@dt.com,user66
user67@dt.com,user67
user68@dt.com,user68
user69@dt.com,user69
user70@dt.com,user70

C:\apache-jmeter-3.0\bin>_
```

Figure 11-45. Users list VM01



```
C:\apache-jmeter-3.0\bin>type continue-shopping.csv
user71@dt.com,user71
user72@dt.com,user72
user73@dt.com,user73
user74@dt.com,user74
user75@dt.com,user75
user76@dt.com,user76
user77@dt.com,user77
user78@dt.com,user78
user79@dt.com,user79
user80@dt.com,user80

C:\apache-jmeter-3.0\bin>_
```

Figure 11-46. Users list VM02

⁴⁴https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_11/DTContinueShoppingTestPlan.jmx

Before starting the test, Alex configured the listeners to write to a file so that after the test, he could load it into JMeter GUI and review performance results (see Figure 11-47).

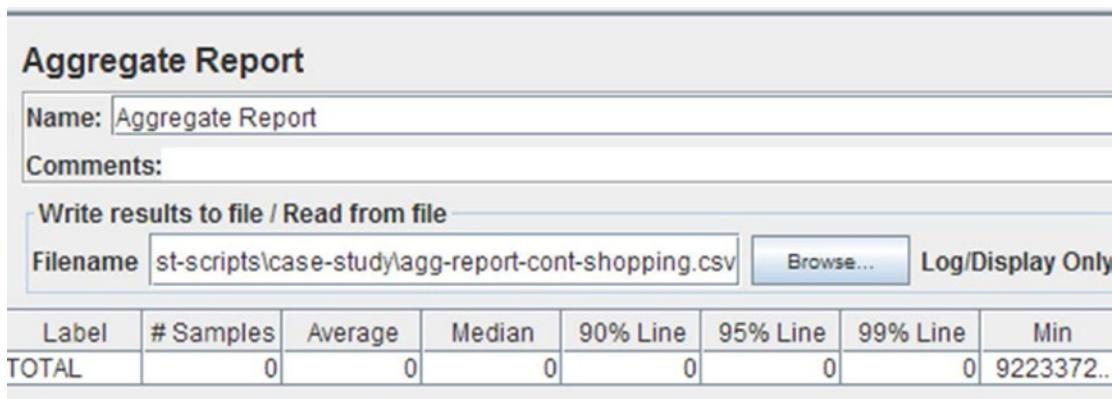


Figure 11-47. Filename aggregate report

Alex also set up the PerfMon Server Agent on the application server and configured it in the JMeter test script so that he could monitor CPU, memory, and network, while executing tests from the slaves (see Figure 11-48).

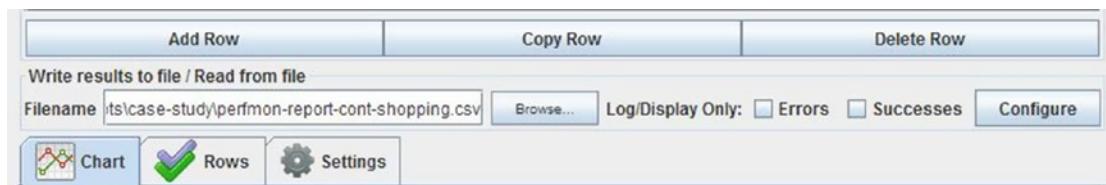


Figure 11-48. Filename PerfMon report

Alex ran the test from the console and showed the output here:

```
C:\>jmeter -n -t DTContinueShoppingTestPlan.jmx -r
Writing log file to: C:\github\jmeter-test-scripts\case-study\jmeter.log
Creating summariser <summary>
Created the tree successfully using DTContinueShoppingTestPlan.jmx
Configuring remote engine: 192.168.1.7
Configuring remote engine: 192.168.1.8
Starting remote engines
Starting the test @ Thu May 18 10:47:34 PDT 2017 (1495129654354)
Remote engines have been started
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary = 216 in 00:01:46 = 2.0/s Avg: 86 Min: 0 Max: 930 Err: 0 (0.00%)
Tidying up remote @ Thu May 18 10:49:23 PDT 2017 (1495129763436)
... end of run
C:\github\jmeter-test-scripts\case-study>
```

The vm01 log's output is as follows.

```
C:\apache-jmeter-3.0\bin>jmeter-server
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter-server.log
Created remote object: UnicastServerRef [liveRef: [endpoint:[192.168.1.7:53129]
(local),objID:[-34fbde0b:15c1c9dba9e:-ffff, 770769319327732
0207]]]
Starting the test on host 192.168.1.7 @ Thu May 18 10:47:09 PDT 2017 (1495129629730)
Finished the test on host 192.168.1.7 @ Thu May 18 10:48:56 PDT 2017 (1495129736089)
```

The vm02 log's output is as follows:

```
C:\>jmeter-server
Could not find ApacheJmeter_core.jar ...
... Trying JMETER_HOME=..
Found ApacheJMeter_core.jar
Writing log file to: C:\apache-jmeter-3.0\bin\jmeter-server.log
Created remote object: UnicastServerRef [liveRef: [endpoint:[192.168.1.8:49568](local),objID
:[19039a4:15c1c9de321:-ffff, 910415358497333192]]]
Starting the test on host 192.168.1.8 @ Thu May 18 10:47:09 PDT 2017 (1495129629182)
Finished the test on host 192.168.1.8 @ Thu May 18 10:48:55 PDT 2017 (1495129735393)
```

The aggregate report captures the data of both the slaves. Alex has 10 threads configured in the test, and with two slaves, that becomes 20 (see Figures 11-49 and 11-50).

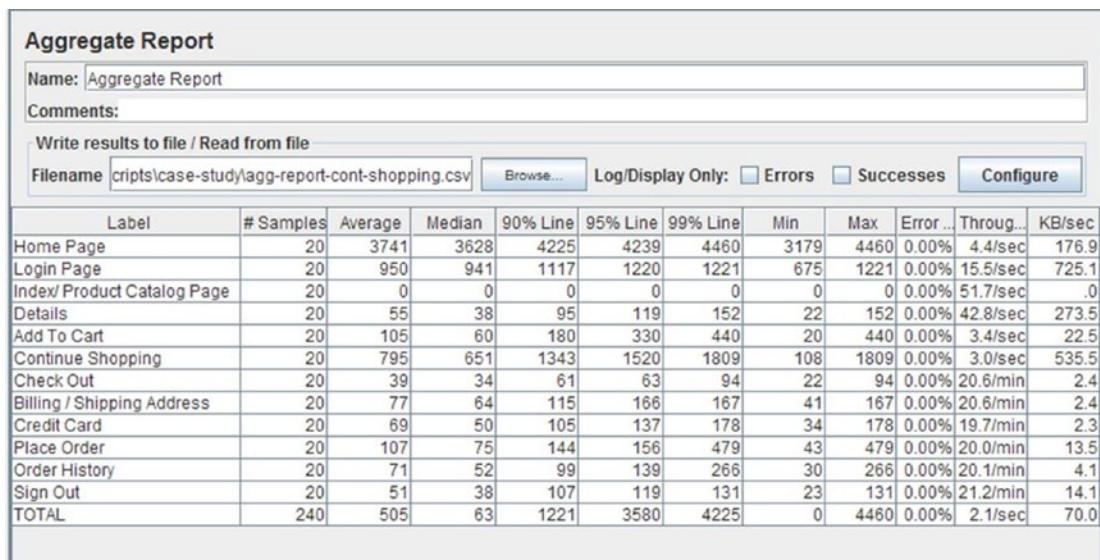


Figure 11-49. Aggregate report distributed load

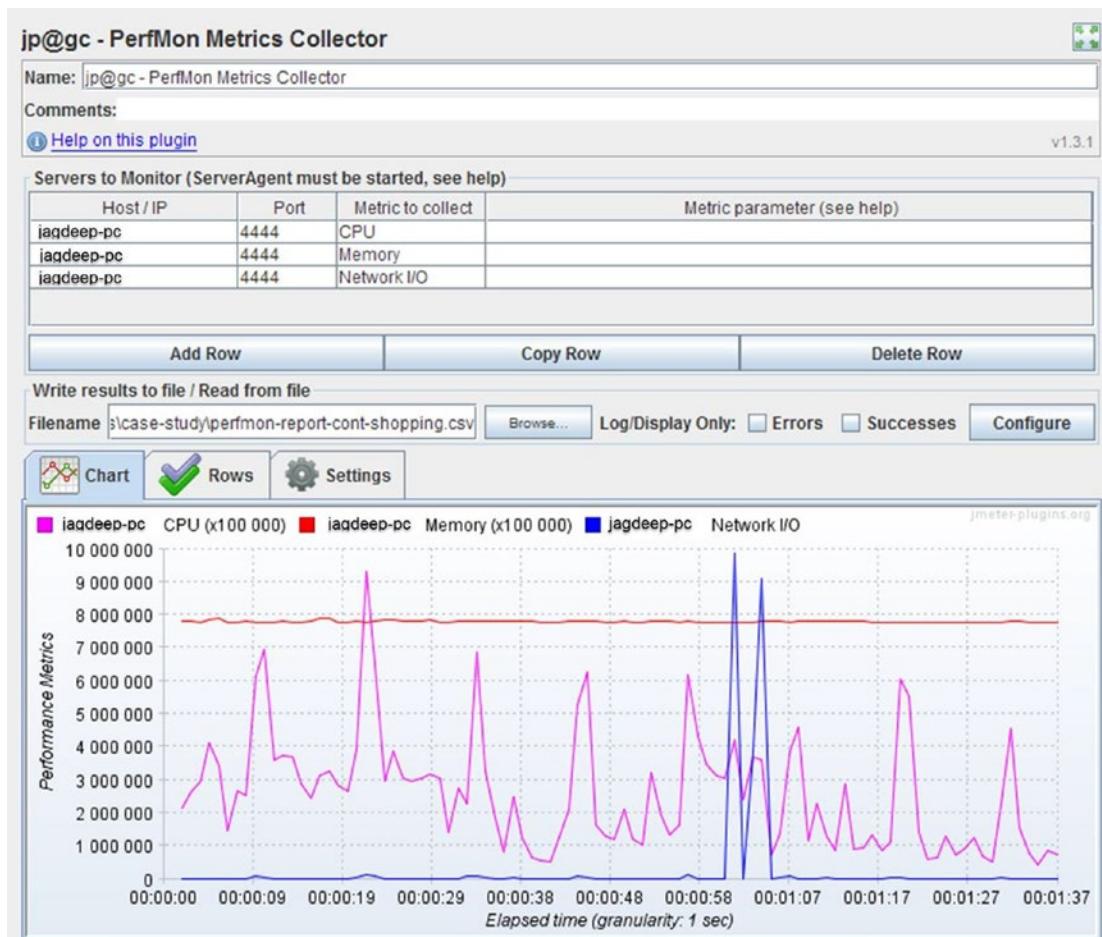


Figure 11-50. PerfMon report distributed load (see color image in source code file)

Tip Use a distributed testing environment to generate more load on the application under test.

Performance Testing and Tuning Cycle

The team iterated through a few rounds of performance testing, followed by performance enhancements and tuning. After a few rounds, the performance improved and the web application met the established performance goals.

Outcome

After the new version of Digital Toys Inc. web application was deployed, Bob reviewed the analytics report and was pleased to see a decrease in response time and an increase in user engagement.

After one month, in the senior management meeting, the Customer Support Manager reported a decrease in support calls. Nancy, the CFO, reported an increase in revenue.

Martin, the CEO, was very pleased with the overall teamwork and the outcome of the performance testing and tuning project. He thanked the team and recognized the importance of performance testing and tuning. He allocated a budget and made performance testing and tuning mandatory for every production release.

Conclusion

In this chapter, you learned how Digital Toys Inc. solved its business problem by using JMeter for performance test automation. You also learned how Alex planned from start to end and involved all the stakeholders along the way. In the next chapter, you learn about the APDEX index and its use in analyzing performance test results. You will also generate Performance Dashboards, which will be useful in understanding load patterns. You will also learn about JVisualVM, which comes with Java by default.

CHAPTER 12



Performance Dashboard

This chapter discusses how to customize and generate the *Performance Dashboard*.

At the end of this chapter, you will have a good idea of the Performance Dashboard, including how it should be generated and how to interpret it.

JMeter 3.0 introduced the concept of the Performance Dashboard, using which you can monitor and analyze the performance of the web application under the test. It consists of a standard set of performance metrics and graphs. These can be customized to suit specific needs.

APDEX

APDEX stands for Application Performance Index.¹ It calculates a user satisfaction score, taking into account user satisfaction levels.

The users are *satisfied* if the response times are below a certain threshold. The users who are *tolerating* are unhappy with the web site performance but continue to use it grudgingly. The users who are *frustrated* have exhausted their patience and abandon the web site. Figure 12-1 shows the formula and Table 12-1 shows the APDEX levels.

$$Apdex_t = \frac{SatisfiedCount + \frac{ToleratingCount}{2}}{TotalSamples}$$

Figure 12-1. APDEX user satisfaction formula

Table 12-1. APDEX Levels

Satisfied	response time < t	Users are satisfied
Tolerating	response time $\geq t$ and $< 4t$	Users are tolerating
Frustrated	response time $> 4t$	Users are frustrated

Configuration

JMeter 3.0 has provided several properties to configure the Performance Dashboard. These properties are present in the `reportgenerator.properties` file and they have to be copied to the `user.properties` file. These properties have to be updated as per the requirements for generating graphs.

¹<https://en.wikipedia.org/wiki/Apdex>

The following sections provide the details to generate and customize the Performance Dashboard.

JMeter Properties

The following properties need to be set to true to allow the generation of graphs under the `jmeter.properties` files.

```
jmeter.save.saveservice.label=true
jmeter.save.saveservice.response_code=true
jmeter.save.saveservice.response_message=true
jmeter.save.saveservice.successful=true
jmeter.save.saveservice.thread_name=true
jmeter.save.saveservice.thread_counts=true
jmeter.save.saveservice.latency=true
jmeter.save.saveservice.bytes=true
.
.
.
# Timestamp format - this only affects CSV output files
# legitimate values: none, ms, or a format suitable for SimpleDateFormat
jmeter.save.saveservice.timestamp_format=ms
jmeter.save.saveservice.timestamp_format=yyyy/MM/dd HH:mm:ss.SSS
```

We can generate the dashboard in a single step via the command line. Alternatively, the test is run and the output is captured to a `.csv` file, which is then used to generate the Performance Dashboard.

APDEX

The following properties under the `reportgenerator.properties` file are used to configure the satisfaction and tolerance thresholds.

```
# Sets the satisfaction threshold for the APDEX calculation (in milliseconds).
jmeter.reportgenerator.apdex_satisfied_threshold=500
# Sets the tolerance threshold for the APDEX calculation (in milliseconds).
jmeter.reportgenerator.apdex tolerated_threshold=1500
```

Global Graph Properties

The following properties are used to set the granularity of the generated graphs.

```
# Regular Expression which Indicates which samples to keep for graphs and statistics
generation.
# Empty value means no filtering
#jmeter.reportgenerator.sample_filter=

# Sets the size of the sliding window used by percentile evaluation.
# Caution : higher value provides a better accuracy but needs more memory.
#jmeter.reportgenerator.statistic_window = 200000
```

```
# Configure this property to change the report title
#jmeter.reportgenerator.report_title=Apache JMeter Dashboard

# Defines the overall granularity for over time graphs
jmeter.reportgenerator.overall_granularity=60000

# Sets the destination directory for generated html pages.
# This will be overridden by the command line option -o
#jmeter.reportgenerator.exporter.html.property.output_dir=report-output
```

You can customize the title of the Performance Dashboard by configuring the `report_title` property. You can also configure granularity of generated graphs by updating the `overall_granularity` property. The samplers that will appear in the graphs can be filtered by updating the `sample_filter` property with a suitable regular expression.

We can use the `-o` option to specify the output directory for the reports. This can be overridden by configuring the `output_dir` property in the `user.properties` file.

Specific Graph Properties

There are a number of graphs that are generated as a part of the Performance Dashboard. The Response Time Vs Request graph is shown here.

```
# Response Time Vs Request graph definition
jmeter.reportgenerator.graph.responseTimeVsRequest.classname=org.apache.jmeter.report.processor.graph.impl.ResponseTimeVSRequestGraphConsumer

jmeter.reportgenerator.graph.responseTimeVsRequest.title=Response Time Vs Request

jmeter.reportgenerator.graph.responseTimeVsRequest.exclude_controllers=true

jmeter.reportgenerator.graph.responseTimeVsRequest.property.set_granularity=${jmeter.reportgenerator.overall_granularity}
```

It is a best practice to have the names of the transactions appear in the graph. To achieve, that do the following:

1. Name the Transaction Controller appropriately.
2. Enable the **Generate Parent Sample** checkbox (see Figure 12-2).

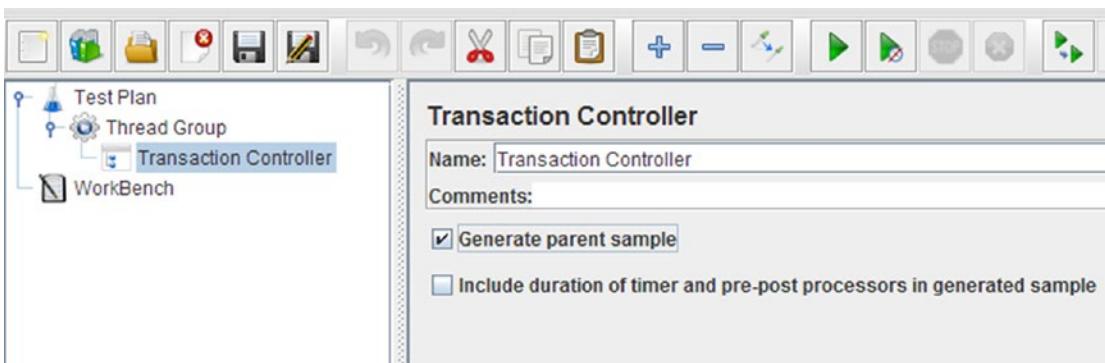


Figure 12-2. Generate parent sample

3. Configure the following property as false:

```
jmeter.reportgenerator.graph.responseTimeVsRequest .exclude_controllers=false
```

The data points are aggregated based on the *granularity*, which can be configured using the following:

```
jmeter.reportgenerator.graph.responseTimeVsRequest.property.set_granularity=${jmeter.reportgenerator.overall_granularity}
```

Generating Graphs

To generate the dashboard in a single step, execute the following command in the CMD prompt.

```
jmeter -n -t <TestName.jmx> -l <logFileName>.jtl -e -o <path of the directory which is used to store Performance Dashboard>
```

For example:

```
jmeter -n -t DTFirstOrderTestPlan.jmx -l dt-firstordertest-plan.jtl -e -o C:\tmp\performance-dashboard\
```

Make sure that the output directory exists and is writable.

Tip Review the `jmeter.log` file for errors.

Performance Dashboard Graphs

The Performance Dashboard serves two purposes:

- Confirms that the test ran as planned with the required user activity and without any errors.
- Obtains the performance metrics.

Let's look at an example to generate the Performance Dashboard.

Follow these steps or download `DTPerformanceDashBoardTestPlan.jmx`.²

1. Start JMeter.
2. Go to File ► Open and select the `DTFirstOrderTestPlan.jmx` file.
3. Click on Test Plan and update it with a meaningful name, such as Performance Dashboard Test.
4. Click on Thread Group, then configure **Number of Threads (users)** as 100 and **Ramp-Up Period (in seconds)** as 100.
5. Save the test plan as `DTPerformanceDashBoardTestPlan.jmx`.
6. Start JVisualVM from the `$JAVA_HOME/bin` directory to gather CPU and memory usage of the web application. This will help in correlating the data with the Performance Dashboard.
7. Execute the following command in the CMD prompt.
`jmeter -n -t DTPerformanceDashBoardTestPlan.jmx -l dt-performance-dashboard-test-plan.jtl -e -o C:\tmp\performance-dashboard\`
8. To open the Performance Dashboard, navigate to the directory `C:\tmp\performance-dashboard\` and open the `index.html` file.

The results derived from the Performance Dashboard are shown here.

The following table shows that the APDEX number is .97 (see Figure 12-3), which indicates high user satisfaction.

APDEX (Application Performance Index)			
Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.990	500 ms	1 sec 500 ms	Total
0.970	500 ms	1 sec 500 ms	Order History
0.979	500 ms	1 sec 500 ms	Details
0.981	500 ms	1 sec 500 ms	Credit Card
0.984	500 ms	1 sec 500 ms	Billing / Shipping Address
0.992	500 ms	1 sec 500 ms	Place Order
0.993	500 ms	1 sec 500 ms	Add To Cart
0.996	500 ms	1 sec 500 ms	Check Out
1.000	500 ms	1 sec 500 ms	Login Page
1.000	500 ms	1 sec 500 ms	Index/ Product Catalog Page
1.000	500 ms	1 sec 500 ms	Home Page
1.000	500 ms	1 sec 500 ms	Sign Out

Figure 12-3. APDEX number

²https://github.com/Apress/pro-apache-jmeter/blob/master/Matam_Ch_12/DTPerformanceDashBoardTestPlan.jmx

The Response Time Over Time graph illustrates how the response time changes over time. It draws our attention to times when the response time was out of acceptable limits. Any such discrepancies should be investigated by the team. From Figure 12-4, it is noted that the response time spiked twice: once at 23:33:00 and once at 23:41:00 (see Figure 12-4).

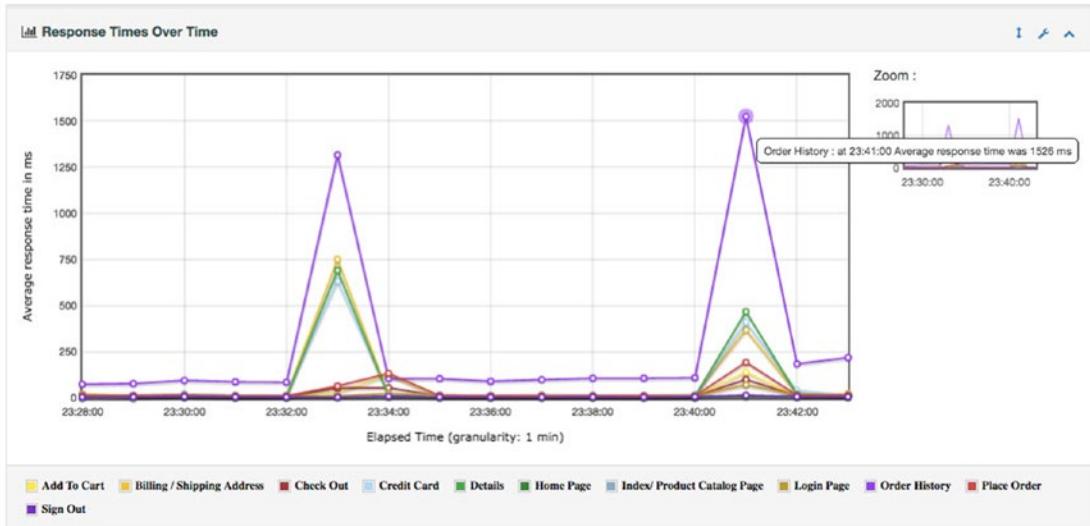


Figure 12-4. Response time over time (see color image in source code file)

The reason for the spikes can be inferred from the JVisualVM graph. You can see that the CPU also spiked at the same time. The spike at 23:33:00 was due to general reasons and the spike at 23:41:00 can be attributed to a garbage collection (GC) activity at the same time (see Figure 12-5).

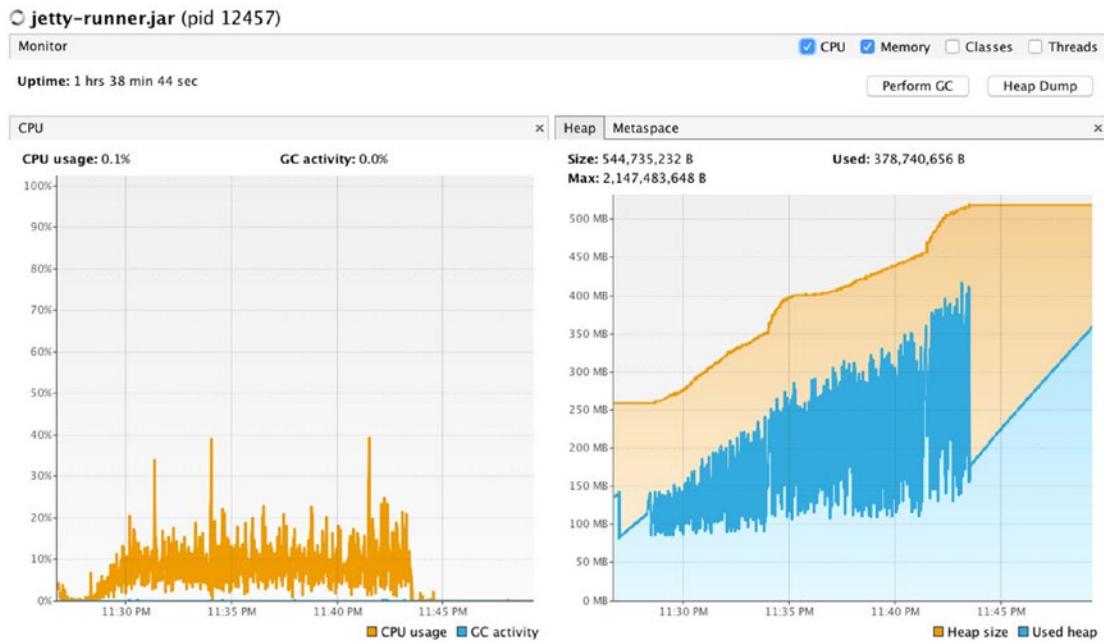


Figure 12-5. JVisualVM statistics

The Bytes Throughput Over Time graph shows data transfer during the request and response. This is especially useful when your web application is data intensive. For example, for a photo or a file sharing application (see Figure 12-6).

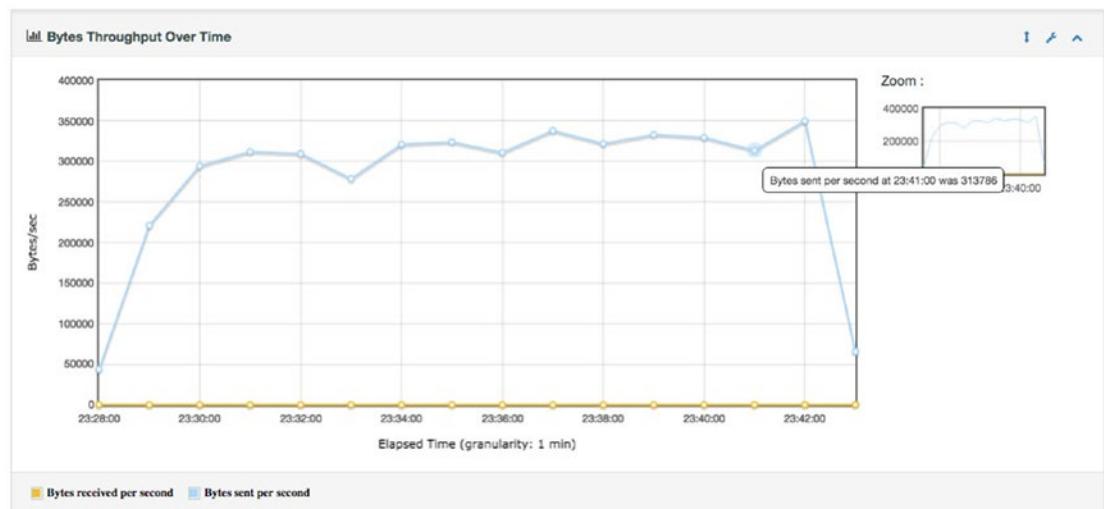


Figure 12-6. Bytes throughput over time

The Hits Per Second graph shows user activity on the web site. By looking at this graph, you can be assured that the web application was subjected to the desired load (see Figure 12-7).

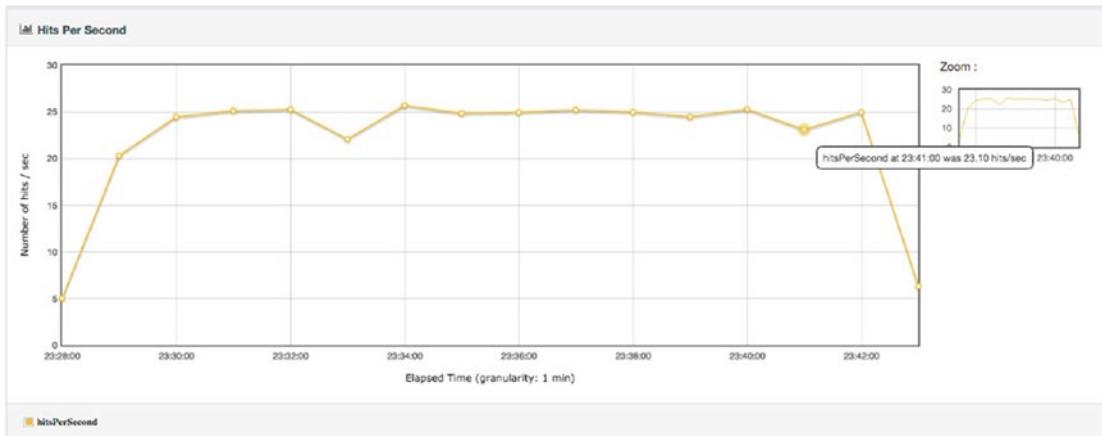


Figure 12-7. Hits per second

The Codes Per Second graph shows HTTP response codes returned from the server. This graph is useful to confirm that there were no excessive errors while load testing the web application (see Figure 12-8).

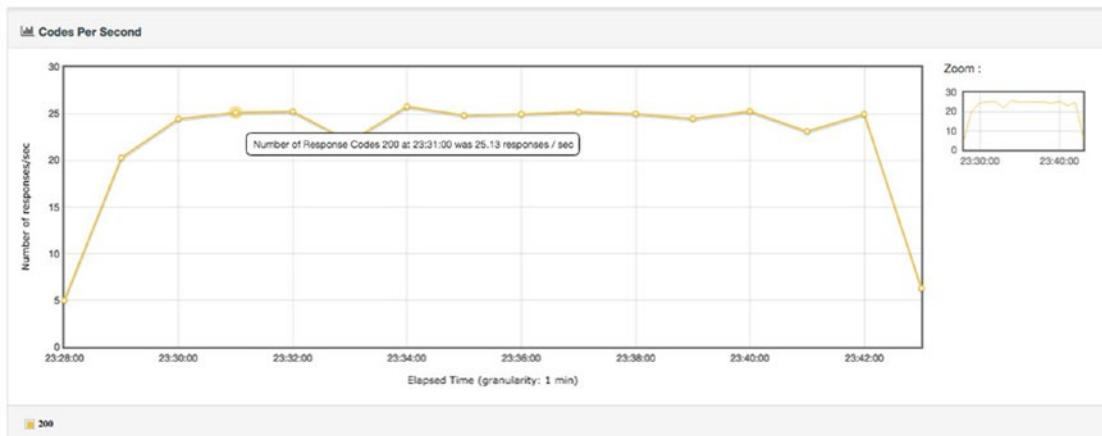


Figure 12-8. Codes per second

The Transactions Per Second graph shows the number of transactions for each use-case. This is useful to confirm that the web application is getting the proper load with the desired mix of transactions (see Figure 12-9).

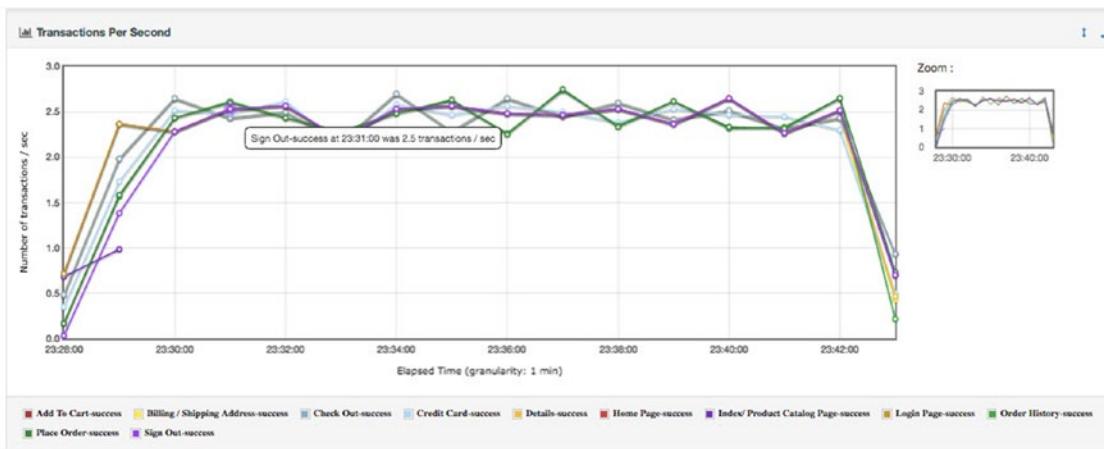


Figure 12-9. Transactions per second (see color image in source code file)

The Response Time Percentiles graph shows the response time deviation (refer APDEX) (see Figure 12-10).

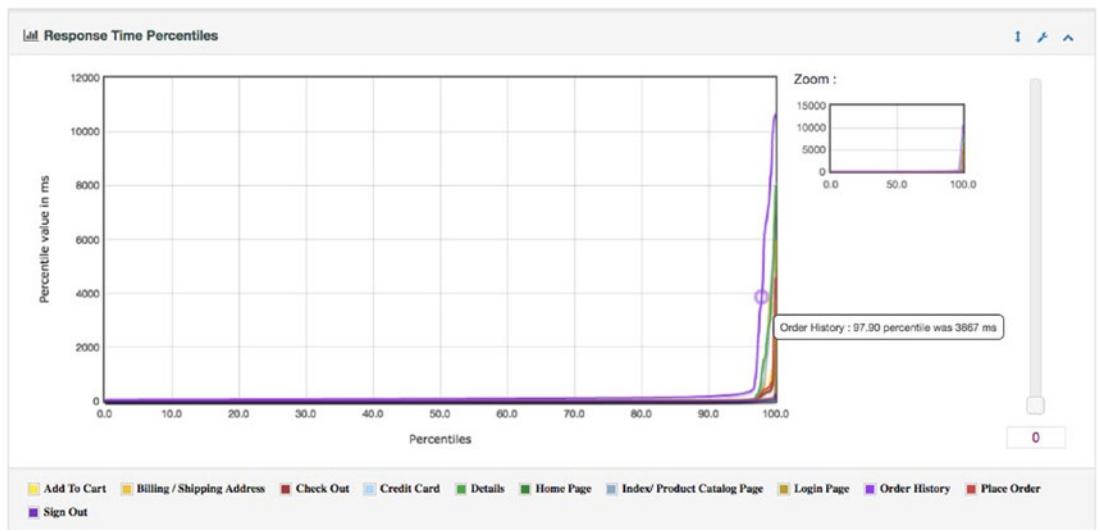


Figure 12-10. Response time percentiles (see color image in source code file)

The Active Threads Over Time graph shows that the number of active users (Threads) are as planned (see Figure 12-11).

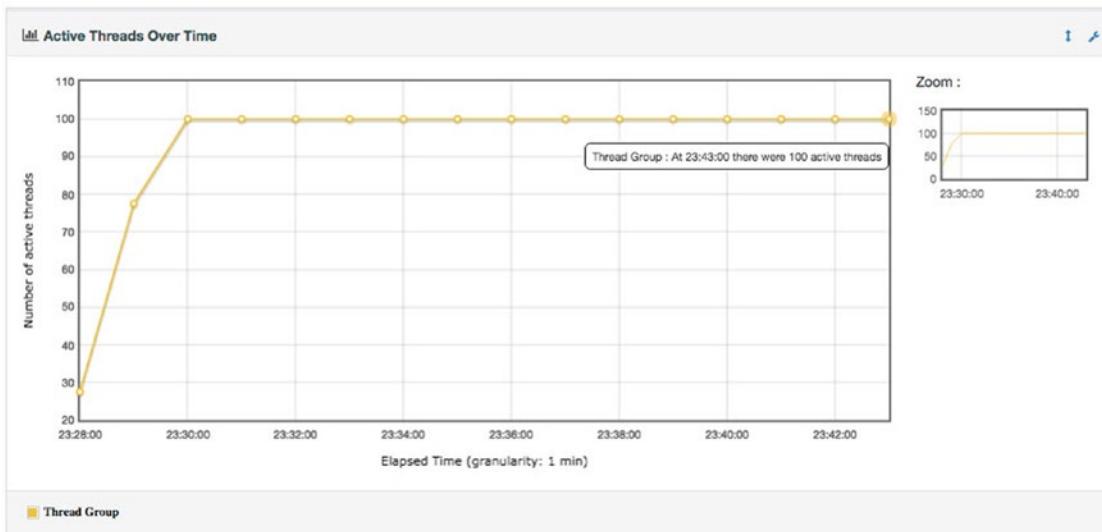


Figure 12-11. Active threads over time

The Time Vs Threads graph shows the number of users (Threads) engaged in a particular transaction over time. In the graph, you can select transactions to compare (see Figure 12-12).

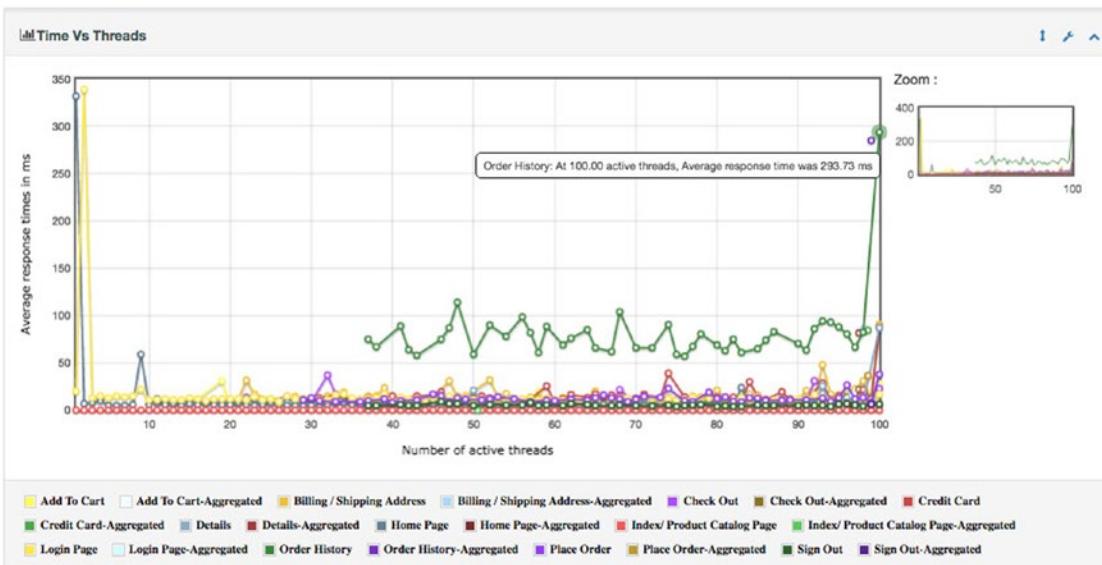


Figure 12-12. Time vs threads (see color image in source code file)

The Response Time Distribution graph shows the response time distribution to identify outliers (see Figure 12-13).

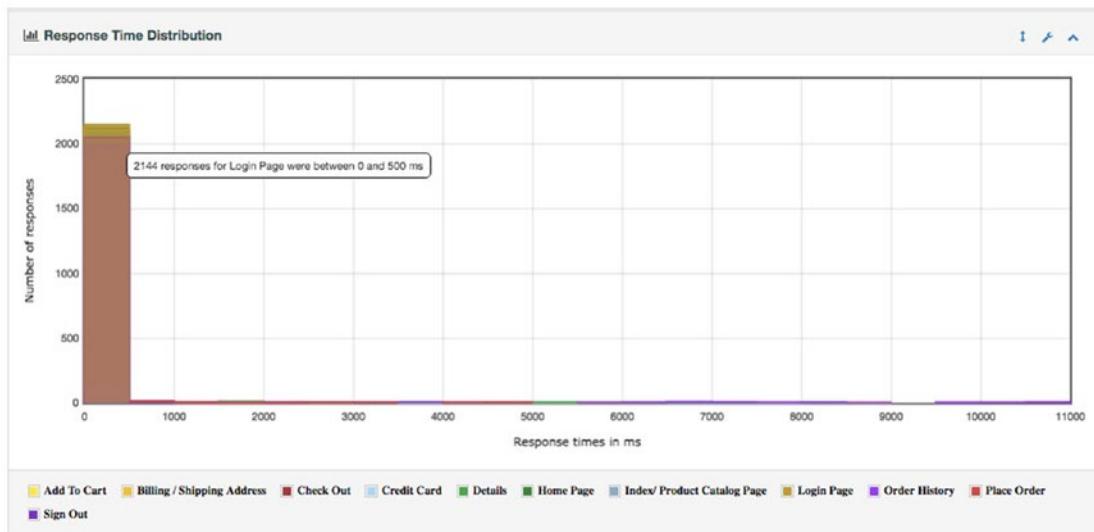


Figure 12-13. Response time distribution

Conclusion

In this chapter, you learned to configure and generate the Performance Dashboard. You also learned to interpret the results.

CHAPTER 13



Appendix A: Setting Up JMeter

In the following sections we describe the installation of Java Development Kit (JDK) and JMeter. We use the JMeter executable to start JMeter.

Note Readers who are already familiar with JMeter setup can start directly with the first chapter.

MacOSX

The following section explains how to install Java and JMeter on MacOSX. If you are using Windows or Linux, skip to the appropriate section.

Download JDK

Download the Java Development Kit (JDK) from Oracle's web site.¹

JMeter requires Java Runtime Environment (JRE) to run. But for some of the advanced uses, we recommend installing a complete development environment. Pick the latest JDK version for MacOSX. This is usually found on the web page by name (`jdk-8u60-macosx-x64.dmg`). Double-click it and follow the instructions on the screen (see Figures 13-1 and 13-2).



Figure 13-1. Launch the installer

¹<http://java.oracle.com>



Figure 13-2. Accept the installer defaults to install JDK

Install JDK

Follow the instructions and complete the installation. To verify the installation of the Java runtime, run the following command in the terminal window.

```
$ java -version
java version "1.8.0_60" Java(TM)
SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

Set Up the Environment Variable

To set the `JAVA_HOME` environment variable, run the following commands in the terminal window.

Open `~/.bash_profile` in your favorite editor.

```
$ vim ~/.bash_profile
```

Or

```
$ vim ~/.profile
```

Add the following line to the file:

```
export JAVA_HOME=$(/usr/libexec/java_home)
```

Log out from the current terminal window and open a new terminal window to pick up the new changes. Or, you can run the following command in the current terminal window.

```
$ source ~/.bash_profile or source ~/.profile
```

Issue the following command to verify that `JAVA_HOME` has been set under the environment variables.

```
$ echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home
```

Issue the following command to verify that JDK has been installed properly and you have Performance Dashboard environment variable. (This verifies the Java compiler.)

```
$ javac -version
javac 1.8.0_60
```

Download JMeter

Download JMeter from the Apache web site.²

Pick the latest version of JMeter; as of now, the latest JMeter available is `apache-jmeter-3.0.tgz`.

Set Up JMeter

Setting up JMeter just involves extracting the JMeter binaries into the user-defined location on the user machine. Assuming the downloaded JMeter binaries are in your `Downloads` folder, run the following command in the terminal window.

```
$ tar -xf ~/Downloads/apache-jmeter-3.0.tgz -C /usr/local/
```

Issue the following command in your terminal window to verify that you have extracted the JMeter binaries correctly.

```
$ cd /usr/local/apache-jmeter-3.0/
$ ls -1p
LICENSE
NOTICE
README
bin/
docs/
extras/
lib/
licenses/
printable_docs/
```

It is good practice to set up the `JMETER_HOME` environment variable and add the following lines to the PATH. Open `~/.bash_profile` or `~/.profile` to your favorite editor.

```
$ vim ~/.bash_profile
```

Or

```
$ vim ~/.profile
```

²<https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-3.0.tgz>

And add the following lines.

```
export JMETER_HOME="/usr/local/apache-jmeter-3.0"
export PATH="$PATH:$JMETER_HOME/bin"
```

Log out from the current terminal window and open a new terminal window to pick up the changes. Or you can run the following command in the current terminal window.

```
$ source ~/.bash_profile or source ~/.profile
```

Now that you have set up JMeter, you can start JMeter by running the startup script. Run the following command in the terminal window (see Figure 13-3).

```
$ jmeter
```

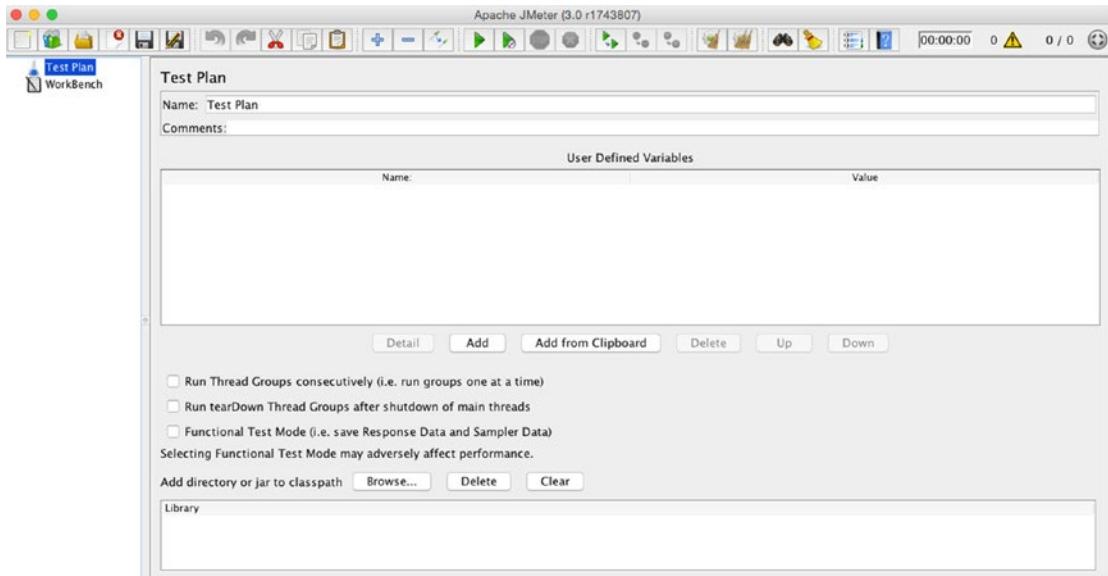


Figure 13-3. JMeter GUI

Windows

The following section shows you how to install Java and JMeter on Windows 7. If you are using Mac OSX or Linux, skip to the appropriate section.

Download JDK

Download the Java Development Kit (JDK) from Oracle's web site.³

³<http://java.oracle.com>

JMeter requires just the Java Runtime Environment (JRE) to run. But for some of the advanced uses, we recommend installing a complete development environment. Pick the latest JDK for Windows. This is usually found on the web page by name. They are called `jdk-8u60-windows-x64.exe` for 64-bit and `jdk-8u60windows-i586.exe` for 32-bit. Depending on your machine configuration, download the required JDK.

Install JDK

Double-click the executable to launch the installer and follow the instructions (see Figures 13-4, 13-5, and 13-6).



Figure 13-4. Installer step 1



Figure 13-5. Installer step 2

**Figure 13-6.** Installer step 3

To verify the installation of Java runtime, run the following command on the Windows command prompt.

```
C:\> java -version
java version "1.8.0_60" Java(TM)
SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
C:\>
```

Set Up the Environment Variable

To set the `JAVA_HOME` environment variable, run the following commands on the Windows command prompt.

```
C:\> setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_60"
C:\> setx PATH "%PATH%;%JAVA_HOME%\bin";
```

Close the Windows command prompt.

Issue the following command to verify that `JAVA_HOME` has been set under the environment variables.

```
C:\> echo %JAVA_HOME%
C:\Program Files\Java\jdk1.8.0_60
```

Issue the following command to verify that JDK has been installed properly and you have set the correct environment variable. (This verifies the Java compiler.)

```
C:\> javac -version
javac 1.8.0_60
```

Download JMeter

Download JMeter from the Apache web site.⁴

Pick the latest version of JMeter; as of the writing of this book, the latest JMeter available is apachejmeter-3.0.zip.

Set Up JMeter

Setting up JMeter involves extracting the JMeter binaries at a user-defined location. Navigate to your Downloads folder where you have downloaded the binaries, right-click the file, choose Extract All, and then follow the instructions on the screen.

Once the extraction is complete, navigate to the JMeter folder to see the contents. The results will resemble Figure 13-7.

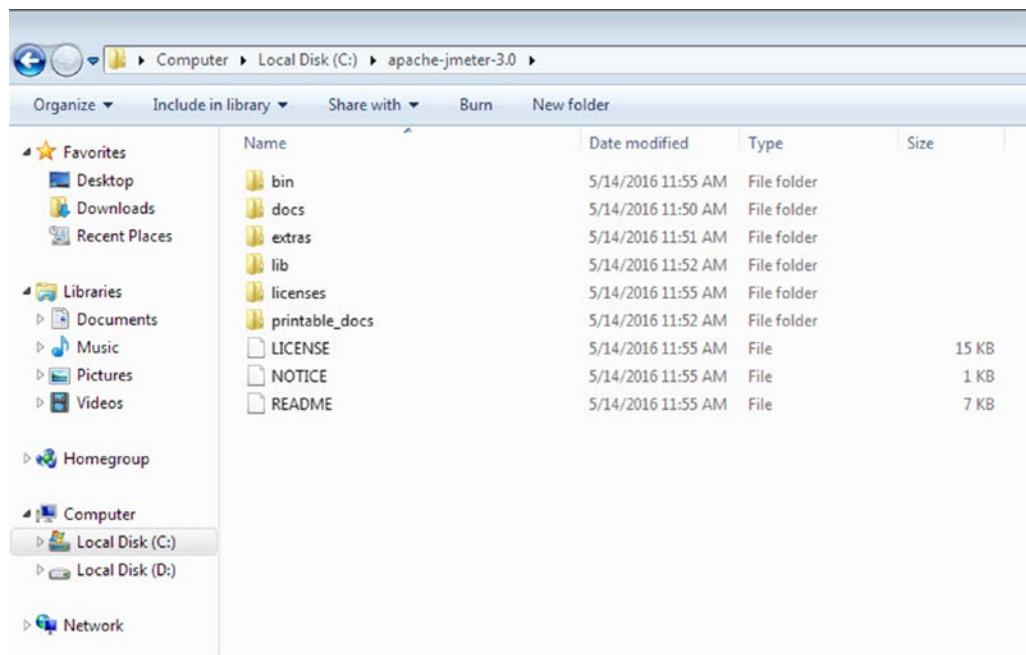


Figure 13-7. JMeter contents

It is a good practice to set up the `JMETER_HOME` environment variable and add this into the PATH.

```
C:\> setx JMETER_HOME "C:\apache-jmeter-3.0"
C:\> setx PATH "%PATH%;%JMETER_HOME%\bin";
```

⁴<https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-3.0.zip>

To start Jmeter, run the following command on your Windows command prompt.

```
C:\> cd C:\apache-jmeter-3.0\bin
C:\apache-jmeter-3.0\bin> jmeter.bat
```

You will see something similar to Figure 13-8.

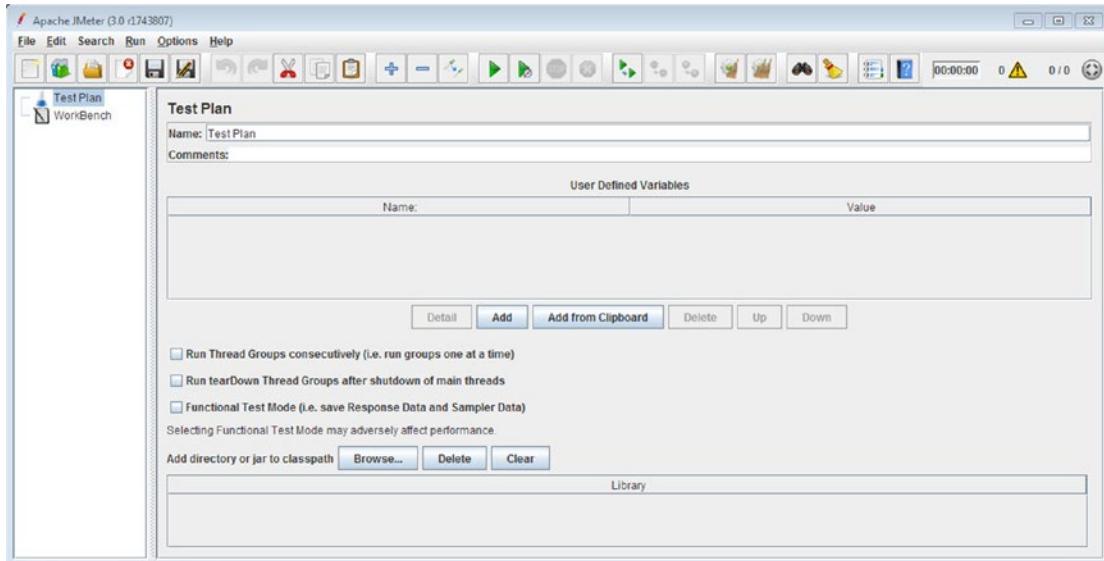


Figure 13-8. JMeter GUI

Linux

The following section explains how to install Java and JMeter on Linux. If you are using Mac OSX or Windows, skip to the appropriate section.

Install JDK

On Fedora, Oracle Linux, and Red Hat Enterprise Linux, open the terminal window and issue the following command.

```
$ su -c "yum install java-1.8.0-openjdk"
```

Open JDK Java 8 was released in March 2014 and has been made into official Ubuntu repositories for 14.10 Utopic and higher. For Ubuntu 14.04, Ubuntu 12.04, and Linux Mint 17, you have to install it from PPA. It's available in Ubuntu Software Center for Ubuntu 14.10 and Ubuntu 15.04.

Open the terminal window in Ubuntu and issue the following command.

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk
```

For more information on JDK installation, check the JDK installation web site.⁵

To verify the installation of Java runtime, run the following command in the terminal window.

```
$ java -version
openjdk version "1.8.0_45-internal"
OpenJDK Runtime Environment (build 1.8.0_45-internal-b14)
OpenJDK 64-Bit Server VM (build 25.45-b02, mixed mode)
```

Set Up the Environment Variable

To set the `JAVA_HOME` environment variable, run the following commands in the terminal window.

Open `~/.bashrc` in your favorite editor.

```
$ gedit ~/.bashrc
```

Add the following lines.

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-adm64
PATH=$PATH:$JAVA_HOME/bin
```

Log out from the current terminal window and open a new terminal window to pick up the changes. Or you can run the following command in the current terminal window.

```
$ source ~/.bashrc
```

Issue the following command to verify that `JAVA_HOME` has been set under the environment variables.

```
$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-adm64
```

Issue the following command to verify that JDK has been installed properly and you have set the correct environment variable. (This verifies the Java compiler.)

```
$ javac -version
javac 1.8.0_45-internal
```

Download JMeter

Download JMeter from the Apache web site.⁶

Pick the latest version of JMeter; as of the writing of this book, the latest JMeter available is `apache-jmeter-3.0.tgz`.

⁵<http://openjdk.java.net/install/>

⁶<https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-3.0.tgz>

Set Up JMeter

Setting up JMeter just involves extracting the JMeter binaries at a user-defined location. Assuming the downloaded JMeter binaries are in your Downloads folder, run the following command in the terminal window.

```
$ sudo tar -xf ~/Downloads/apache-jmeter-3.0.tgz -C /usr/ local/
```

Issue the following command in your terminal window to verify that you have extracted JMeter binaries correctly.

```
$ cd /usr/local/apache-jmeter-3.0/
$ ls -1p
bin/
docs/
extras/
lib/
LICENSE
licenses/
NOTICE
printable_docs/
README
```

It is good practice to set up the `JMETER_HOME` environment variable and add this into the PATH. Open `~/.bashrc` in your favorite editor and add the following lines.

```
$ gedit ~/.bashrc
```

Add the following lines.

```
export JMETER_HOME="/usr/local/apache-jmeter-3.0"
export PATH=$PATH:$JMETER_HOME/bin
```

Log out from the current terminal window and open a new terminal window to pick up the changes. Or you can run the following command in the current terminal window.

```
$ source ~/.bashrc
```

Now that you have set up JMeter, you can start JMeter by running the JMeter startup script. Run the following command in the terminal window (see Figure 13-9).

```
$ jmeter
```

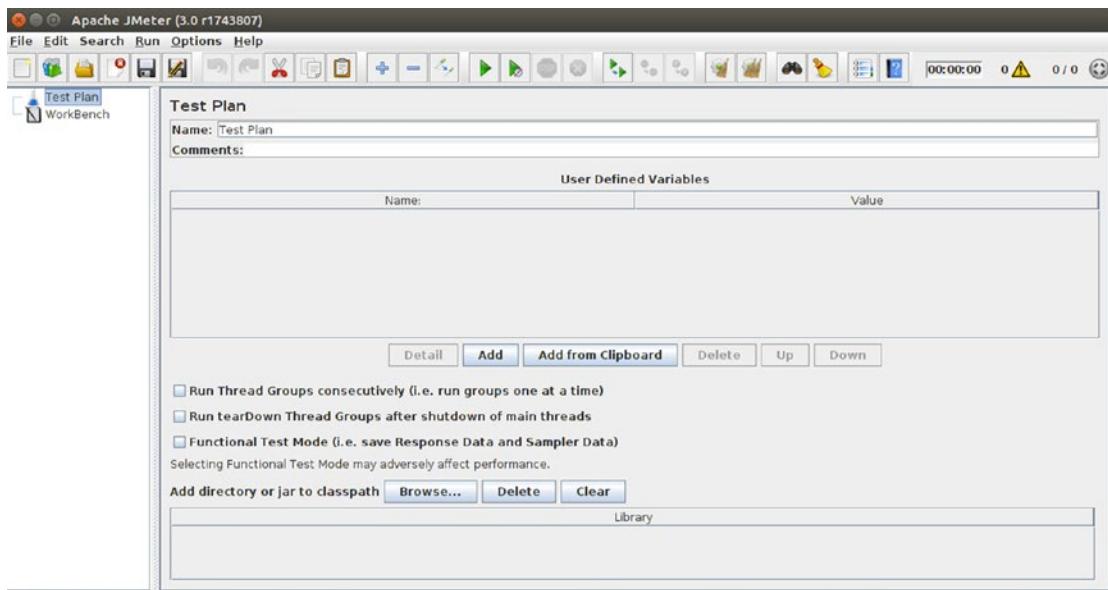


Figure 13-9. JMeter GUI Ubuntu

Note Going forward, we use Windows-based examples, but you can use the same steps on MacOS or any Linux OS.

CHAPTER 14



Appendix B: Setting Up Digital Toys Inc.

We have used the Digital Toys Inc. web application for explaining the examples in this book. We describe setting up this web application in this chapter.

Running Digital Toys Web Application

Clone the GitHub repository¹ or download it as a .zip from the URL.

After extracting the .zip in some location on your machine and listing the files in the extracted folder, you will find these files.

- jetty-runner.jar
- dt-1.0.war
- startDt.sh
- startDt.bat

Use .bat if you are using Windows environment, otherwise use .sh.

Note Make sure that you have JDK 8.0 installed on your machine. The Digital Toys Inc. web application has been compiled using JDK 8.0.

Start the Web Application

Enter the following command in the Windows command prompt.

```
C:\>startDt.bat
```

```
C:\>java -jar jetty-runner.jar dt-1.0.war
```

```
2017-05-15 10:53:10.005:INFO::main: Logging initialized @872ms  
2017-05-15 10:53:10.045:INFO:oejr.Runner:main: Runner
```

¹<https://github.com/Apress/pro-apache-jmeter/tree/master/digital-toys-web-application>

```
2017-05-15 10:53:10.584:INFO:oejs.Server:main: jetty-9.3.7.v20160115
2017-05-15 10:53:52.719:INFO:/:main: No Spring WebApplicationInitializer types detected on
classpath
SESSION_IN_URL not specified.
2017-05-15 10:53:57.182:INFO:/:main: Initializing Spring root WebApplicationContext
SESSION_IN_URL not specified.
Populating states
State: [Alabama, AL] -- Alabama and AL
State: [Alaska, AK] -- Alaska and AK
State: [Arizona, AZ] -- Arizona and AZ
State: [Arkansas, AR] -- Arkansas and AR
State: [California, CA] -- California and CA
State: [Colorado, CO] -- Colorado and CO
State: [Connecticut, CT] -- Connecticut and CT
State: [Delaware, DE] -- Delaware and DE
State: [Florida, FL] -- Florida and FL
State: [Georgia, GA] -- Georgia and GA
State: [Hawaii, HI] -- Hawaii and HI
State: [Idaho, ID] -- Idaho and ID
State: [Illinois, IL] -- Illinois and IL
State: [Indiana, IN] -- Indiana and IN
State: [Iowa, IA] -- Iowa and IA
State: [Kansas, KS] -- Kansas and KS
State: [Kentucky, KY] -- Kentucky and KY
State: [Louisiana, LA] -- Louisiana and LA
State: [Maine, ME] -- Maine and ME
State: [Maryland, MD] -- Maryland and MD
State: [Massachusetts, MA] -- Massachusetts and MA
State: [Michigan, MI] -- Michigan and MI
State: [Minnesota, MN] -- Minnesota and MN
State: [Mississippi, MS] -- Mississippi and MS
State: [Missouri, MO] -- Missouri and MO
State: [Montana, MT] -- Montana and MT
State: [Nebraska, NE] -- Nebraska and NE
State: [Nevada, NV] -- Nevada and NV
State: [New Hampshire, NH] -- New Hampshire and NH
State: [New Jersey, NJ] -- New Jersey and NJ
State: [New Mexico, NM] -- New Mexico and NM
State: [New York, NY] -- New York and NY
State: [North Carolina, NC] -- North Carolina and NC
State: [North Dakota, ND] -- North Dakota and ND
State: [Ohio, OH] -- Ohio and OH
State: [Oklahoma, OK] -- Oklahoma and OK
State: [Oregon, OR] -- Oregon and OR
State: [Pennsylvania, PA] -- Pennsylvania and PA
State: [Rhode Island, RI] -- Rhode Island and RI
State: [South Carolina, SC] -- South Carolina and SC
State: [South Dakota, SD] -- South Dakota and SD
State: [Tennessee, TN] -- Tennessee and TN
State: [Texas, TX] -- Texas and TX
State: [Utah, UT] -- Utah and UT
```

```

State: [Vermont, VT] -- Vermont and VT
State: [Virginia, VA] -- Virginia and VA
State: [Washington, WA] -- Washington and WA
State: [West Virginia, WV] -- West Virginia and WV
State: [Wisconsin, WI] -- Wisconsin and WI
State: [Wyoming, WY] -- Wyoming and WY
Populating products
2017-05-15 10:54:32.277:INFO::main: Initializing Spring FrameworkServlet 'grails'
2017-05-15 10:54:33.567:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@707f70
52{/file:///C:/Users/jjain/AppData/Local/Temp/je
tty-0.0.0-8080-dt-1.0.war-_any-2712378741143744577.dir/webapp/,AVAILABLE}{..}
2017-05-15 10:54:33.751:INFO:oejs.ServerConnector:main: Started ServerConnector@3a2a56f6
{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
2017-05-15 10:54:33.759:INFO:oejs.Server:main: Started @84653ms

```

If you want to start the Digital Toys Inc. web application on a different port, enter this command in the CMD prompt:

```
C:\>java -jar jetty-runner.jar --port 7070 dt-1.0.war
```

```

2017-05-21 17:14:10.317:INFO::main: Logging initialized @387ms
2017-05-21 17:14:10.347:INFO:oejr.Runner:main: Runner
2017-05-21 17:14:10.657:INFO:oejs.Server:main: jetty-9.3.7.v20160115
2017-05-21 17:14:27.002:INFO::main: No Spring WebApplicationInitializer types detected on
classpath
SESSION_IN_URL not specified.
2017-05-21 17:14:28.094:INFO::main: Initializing Spring root WebApplicationContext
SESSION_IN_URL not specified.
Populating products
2017-05-21 17:14:44.414:INFO::main: Initializing Spring FrameworkServlet 'grails'
2017-05-21 17:14:45.304:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@707f70
52{/file:///C:/Users/jjain/A
ppData/Local/Temp/jetty-0.0.0-7070-dt-1.0.war-_any-5504021582514241800.dir/
webapp/,AVAILABLE}{..}
2017-05-21 17:14:45.344:INFO:oejs.ServerConnector:main: Started ServerConnector@6d035815
{HTTP/1.1,[http/1.1]}{0.0.0.0:70 70}
2017-05-21 17:14:45.344:INFO:oejs.Server:main: Started @35417ms
Found 108 products

```

Open the browser and launch <http://localhost:8080/dt>. A home page will be shown as the landing page of the Digital Toys Inc. web application (see Figure 14-1).

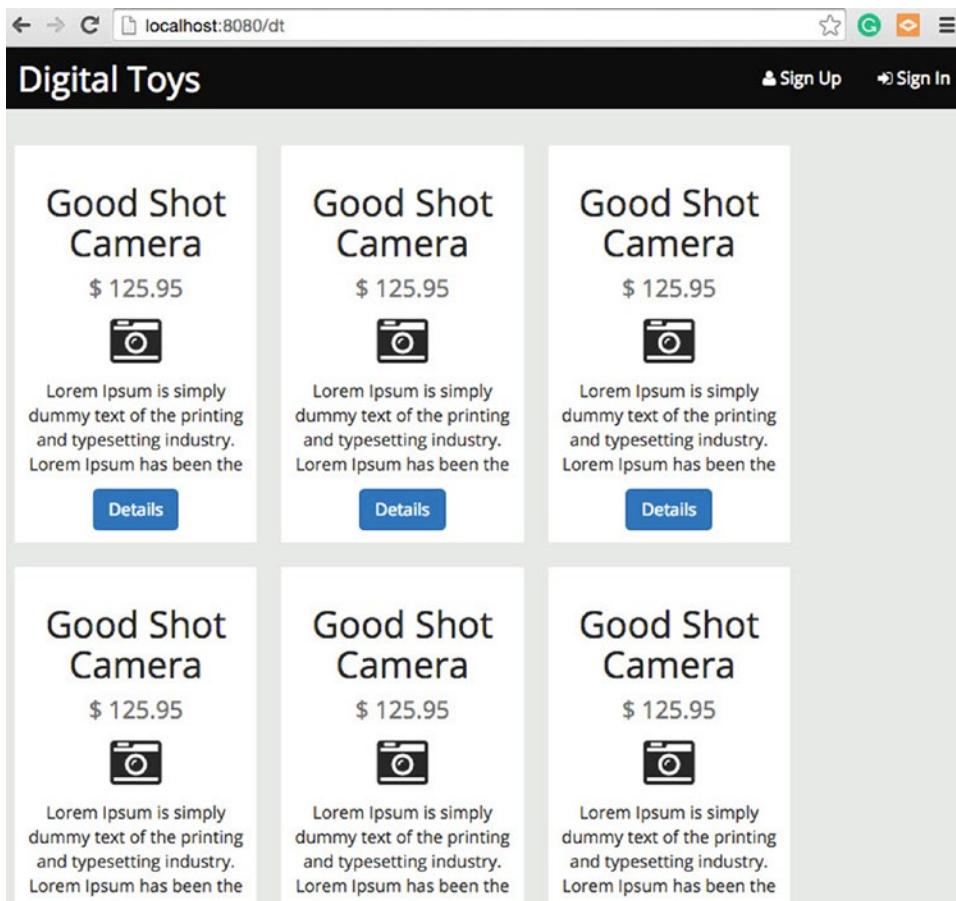


Figure 14-1. Digital Toys landing page

Start with URL Rewriting Enabled

Enter the following command in the CMD prompt:

```
C:\>set SESSION_IN_URL=true
C:\>startDt.bat

C:\>java -jar jetty-runner.jar dt-1.0.war
2017-05-15 11:15:53.013:INFO::main: Logging initialized @518ms
2017-05-15 11:15:53.036:INFO:oejr.Runner:main: Runner
2017-05-15 11:15:53.358:INFO:oejs.Server:main: jetty-9.3.7.v20160115
2017-05-15 11:16:16.901:INFO://:main: No Spring WebApplicationInitializer types detected on classpath
SESSION_IN_URL specified in environment - enabling session in URL if cookies are not enabled.
2017-05-15 11:16:19.121:INFO://:main: Initializing Spring root WebApplicationContext
SESSION_IN_URL specified in environment - enabling session in URL if cookies are not enabled.
Populating products
```

```
2017-05-15 11:16:51.178:INFO:/:main: Initializing Spring FrameworkServlet 'grails'  
2017-05-15 11:16:52.225:INFO:oejsh.ContextHandler:main: Started o.e.j.w.WebAppContext@707f70  
52{/file:///C:/Users/jjain/AppData/Local/Temp/jetty-0.0.0-8080-dt-1.0.war-_any-4592381167383482346.dir/webapp/,AVAILABLE}{...}  
2017-05-15 11:16:52.266:INFO:oejs.ServerConnector:main: Started ServerConnector@552ffffc8  
{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}  
2017-05-15 11:16:52.268:INFO:oejs.Server:main: Started @59776ms
```

Tip Rev This is required for running the preprocessors test plan.

Clean Up

After jetty-runner has started, it will create two files.

Open the CMD prompt and execute the following command:

```
C:\>dir prodDb*  
Volume in drive C has no label.  
Volume Serial Number is DA32-01EE  
  
Directory of C:\  
  
05/15/2017 11:17 AM      124,928 prodDb.h2.db  
05/15/2017 11:16 AM      24,033 prodDb.trace.db  
              2 File(s)      148,961 bytes  
              0 Dir(s)    7,259,287,552 bytes free
```

```
C:\>
```

These files save data. They are useful for deleting files before starting with a fresh chapter.

Index

■ A, B, C

Case study
exclude regular expression, [253, 255–258](#)

■ D, E

Distributed testing, [165, 167–177](#)
batch mode, [173–174](#)
custom implementation mode, [175](#)
DiskStore mode, [174](#)
GUI Mode, [171–172](#)
hold mode, [174](#)
limitations, [177](#)
master-slave, [167, 172, 173](#)
server port, [172, 173](#)
statistical mode, [174](#)
StrippedAsynch mode, [175](#)
StrippedBatch mode, [175](#)
StrippedDiskStore mode, [174](#)
Stripped mode, [174–175](#)
unreachable remote host, [176–177](#)

■ F, G, H, I

First JMeter Test
assertions, [14](#)
configuration elements, [14](#)
controller, [14](#)
executing a single test, [23](#)
GUI mode, [21–22](#)
listener, [14](#)
post-processor, [15](#)
pre-processor, [15](#)
proxy server setting, [24](#)
sampler, [14](#)
start JMeter in server mode, [24](#)
stop/shutdown JMeter, [24](#)
test plan, [13](#)
thread groups, [13](#)
timer, [14](#)

Foundation

why JMeter?, [2](#)
why performance testing?, [1](#)

■ J, K, L, M, N, O

JMeter best practices
CSV data set config, [192–194](#)
HTTP cookie manager, [184](#)
HTTP request defaults, [179–180](#)
JMeter Plugins
JMeter Plugins at Google Code (JP@GC), [211](#)
JMeter Properties, [218](#)
PerfMon, [211–219](#)
ServerAgent, [212–214](#)
JMeter Recipes
FTP-performance
file transfer protocol (FTP), [228–230](#)
JDBC performance testing
JDBC test plan, [224–227](#)
mobile performance testing
characters per second (cps), [234](#)
mobile device, [234, 235](#)
User-Agent, [234](#)
REST-JSON performance testing
HttpClient4, [231](#)
REST, [230–232](#)
SOAP performance testing
SOAP, [237–242](#)
JMeter test plan components
assertions
apply to, [129](#)
assertion results listener, [128](#)
pattern matching rules, [128–130](#)
pattern to test, [130–138](#)
response assertion, [128](#)
response field to test, [129](#)
Controller
ForEach Controller, [85–87](#)
If Controller, [87–92](#)
Interleave Controller, [76–81](#)

■ INDEX

JMeter test plan components (*cont.*)

- Loop Controller, 68–70
- Once Only Controller, 74–76
- Random Controller, 81
- Random Order Controller, 81–82
- Runtime Controller, 70–72
- Simple Controller, 64–65
- Switch Controller, 82–84
- Throughput Controller, 72–74
- Transaction Controller, 65–68

Listener, 138

- Aggregate Report Listener, 150–152
- View Results In Table, 147–149
- View Results Tree, 141–147

post processors

- regular expression extractor, 152–157

pre processors

- HTTP URL re-writing modifier, 61–63

properties and variables, 157–165

JSR223 PostProcessor, 162

properties

user defined variables (UDV), 157, 158,
161–163

variables, 157

sampler

Body Data, 117–119

Browser-compatible headers, 116

Embedded Resources for HTML Files, 124–126

Follow Redirect, 112–114

HTTPClient, 106–109

HTTP Request, 106

Implementation HTTP Request Sampler,
106–109

MD5 hash?, 126–127

MIME Type, 120–122

Option Task, 126–127

proxy server, 123

Redirect Automatically, 110–112

Send Files With the Request, 120–123

Send Parameter With the Request, 117

Source Address, 126

Switching Between Name:Value and Body
Data, 119–120

Use KeepAlive, 115–116

use multipart/form-data for POST, 116

test plan, 35–36

 user defined variables, 41

thread group

 Number of Threads (users), 46–48

 Ramp-Up Period (in seconds), 46–48

 scheduler, 49–53

 thread properties, 43–48

timers

 Constant Delay, 99–100

 Constant Throughput Timer, 101–103

Constant Timer, 93–96

Gaussian Random Timer, 97–99

Random Delay, 99–101

Synchronizing Timer, 103–105

Uniform Random Timer, 99–101

JMeter Test Script Recorder

 Exclude regular expression, 30

 Recording Controller, 28

 URL Patterns To Exclude, 27

 URL Pattern to Include, 27

 WorkBench, 25–27

■ P, Q, R

Performance Dashboard

 active threads over time, 311, 312

 APDEX, 303

 codes per second, 310

 hits per second, 310

 response time distribution, 313

 response time percentiles, 311

 time vs threads, 312

 transactions per second, 310, 311

Performance Testing Primer

 application usage patterns, 10

 capacity test, 6

 load test, 6

 peak load test, 6

 performance criteria, 3

 performance goals, 10

 performance reports, 11

 performance requirements, 10

 performance smoke test, 7

 performance test environment, 9–10

 performance testing strategy, 10–11

 performance test suite, 10

 response time, 3–4

 scalability, 5

 soak test, 6

 spike test, 7

 stress test, 5

 throughput, 4

 utilization, 5

■ S

Setting Up Digital Toys Inc.

 cleanup, 331

 start web application, 327–330

 start with URL rewriting enabled,
 330–331

Setting Up JMeter

 Linux, 322–325

 MacOS, 315–318

 Windows, 318–322

■ T, U, V, W, X, Y, Z

Troubleshooting JMeter

 execute permissions, [197](#)

 HTTP Basic Authentication, [204](#)

 HTTPClient, [199](#)

java.RMI.Remote
Exception, [200](#)
JMeter logging level, [198](#)
log file, [197](#)
Log Viewer, [199, 200](#)
proxy server, [202–203](#)