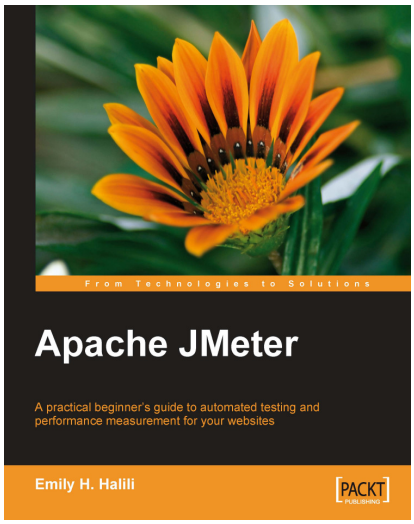
Apache JMeter

Emily H. Halili



Chapter No. 6
"Functional Testing"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.6 "Functional Testing"

A synopsis of the book's content

Information on where to buy this book

About the Author

Emily H. Halili Since graduating in 1998, from California State University in Computer Science, Emily H. Halili has taken numerous roles in the IT/Software industry—namely as Software Engineer, Network Engineer, Lecturer, and Trainer. Currently a QA Engineer in CEO Consultancy-Malaysia with great passion for testing, she has two years of experience in software testing and managing QA activities. She is an experienced manual tester and has practical knowledge of various open-source automation tools and frameworks, including JMeter, Selenium, JProfiler, Badboy, Sahi, Watij, and many more.

My heartfelt thanks to my husband, Duraid Fatouhi, whom without his faith in me, this book may never see the light. To John VanZandt, president of CEO Consultancy, Malaysia – who inspires creativity and comradeship at work. To my colleagues at CEO Consultancy and ex-colleagues, for constantly challenging me with testing tasks and much more. Lastly, but the least, my daughter, Zahraa for inspiring.

Apache JMeter

JMeter is a powerful, easy-to-use, and FREE load-testing tool. Those are my first impressions of JMeter, a testing tool I've recently fallen in love with—not blindly. With this book, I share with you my experience with JMeter.

When I was first assigned to use JMeter to perform testing on a particular web application, I went all out looking for anything on JMeter. Despite plenty of online manuals, article and newsgroup posts, printed or e-books were nowhere to be found. So, when one of the editors of Packtpub approached me with this idea of writing a book on JMeter, I could hear myself saying: "Had there been a book on JMeter, I would have bought one at any cost. Since no one has written any, why not I write one?" After much contemplation and work, here is the result—what you are reading right now.

What This Book Covers

Chapter 1: Automated Testing

The reader who is already automating their tests may want to skip this chapter. It takes a quick look at the need to automate testing and whether automation suits all needs of testing. It provides a quick look at and evaluation of test automation.

Chapter 2: Introduction to JMeter

This chapter is an overview of JMeter, as it takes a glance at its young history, the general look-and-feel of its GUI design, requirements, and its features.

Chapter 3: Getting Started

This chapter serves as a guide to the first-time user on installing and customizing the system environment as they run JMeter for the first time. The installation process will match the purpose of this book. Hence it will skip the more complex setup of the environment. A more complex setup guide is available from the home site of JMeter.

Chapter 4: The Test Plan

This chapter sets out to prepare the reader with the basic knowledge of tools required to successfully create and run tests. It prepares the reader for the next two chapters.

Chapter 5: Load/Performance Testing of Website

This chapter demonstrates the use of the tools in JMeter that support Load or Performance Testing. The walkthroughs are facilitated by illustrations, giving a more descriptive guide to both new and seasoned testers.

Chapter 6: Functional Testing

This chapter demonstrates the use of the tools in JMeter that support Functional or Regression Testing. Little is known of JMeter being used to support this testing approach. As in Chapter 5, the walkthroughs are facilitated by illustrations, giving a more descriptive guide to both new and seasoned testers.

Chapter 7: Advanced Features

This chapter briefly describes other resources that can be tested by using JMeter, i.e. HTTP Server, Database Server, FTP Server, using Regular Expressions, and much more. The reader may want to explore more of JMeter, once he/she has a good understanding of the basics this book covers.

Chapter 8: JMeter and Beyond

This chapter discusses briefly on what more JMeter has and can do for its users. It tells the reader where to go in order to find more information about other elements of JMeter that this book does not have.

6
Functional Testing

JMeter is found to be very useful and convenient in support of functional testing. Although JMeter is known more as a performance testing tool, functional testing elements can be integrated within the Test Plan, which was originally designed to support load testing. Many other load-testing tools provide little or none of this feature, restricting themselves to performance-testing purposes. Besides integrating functional-testing elements along with load-testing elements in the Test Plan, you can also create a Test Plan that runs these exclusively. In other words, aside from creating a Load Test Plan, JMeter also allows you to create a **Functional Test Plan**. This flexibility is certainly resource-efficient for the testing project.

This chapter will give a walkthrough on how to create a Test Plan as we incorporate and/or configure JMeter elements to support functional testing. This chapter assumes that you have successfully gone through Chapter 5, and created a Test Plan for a specific target web server. We will begin the chapter with a quick overview to prepare you with a few expectations about JMeter. Later, we will create a new Test Plan similar to the Test Plan in Chapter 5, only smaller. The Test Plan we will create and run at the end of this chapter will incorporate elements that support functional testing, exclusively.

Preparing for Functional Testing

In this regard, I need to highlight that JMeter does not have a built-in browser, unlike many functional-test tools. It tests on the protocol layer, not the client layer (i.e. JavaScripts, applets, etc.) and it does not render the page for viewing. Although, by default that embedded resources can be downloaded, rendering these in the **Listener | View Results Tree** may not yield a 100% browser-like rendering. In fact, it may not be able to render large HTML files at all. This makes it difficult to test the GUI of an application under testing.

However, to compensate for these shortcomings, JMeter allows the tester to create assertions based on the tags and text of the page as the HTML file is received by the client. With some knowledge of HTML tags, you can test and verify any elements as you would expect them in the browser.

Unlike for a load-testing Test Plan, it is unnecessary to select a specific workload time to perform a functional test. In fact, the application you want to test may even reside locally, with your own machine acting as the "localhost" server for your web application. For this chapter, we will limit ourselves to selected functional aspects of the page that we seek to verify or assert.

Using JMeter Components

We will create a Test Plan in order to demonstrate how we can configure the Test Plan to include functional testing capabilities. The modified Test Plan will include these scenarios:

1. **Create Account**—New Visitor creating an Account
2. **Log in User**—User logging in to an Account

Following these scenarios, we will simulate various entries and form submission as a request to a page is made, while checking the correct page response to these user entries. We will add assertions to the samples following these scenarios to verify the 'correctness' of a requested page. In this manner, we can see if the pages responded correctly to invalid data. For example, we would like to check that the page responded with the correct warning message when a user enters an invalid password, or whether a request returns the correct page.

First of all, we will create a series of test cases following the various user actions in each scenario. The test cases may be designed as follows:

CREATE ACCOUNT

Test Steps	Data	Expected
1 **Go to Home page**.	`www.packtpub.com`	Home page loads and renders with no page error
2 **Click Your Account link** (top right).	User action	1. **Your Account** page loads and renders with no page error. 2. **Logout** link is not found.

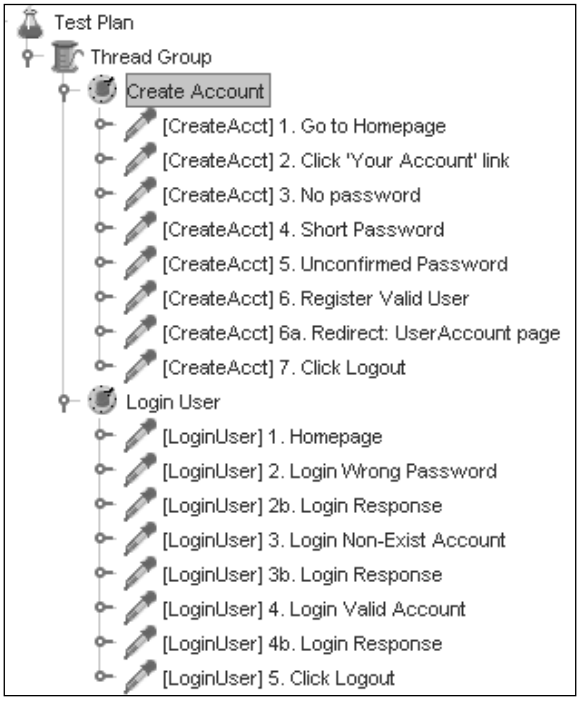
	Test Steps	Data	Expected
3	**No Password**: - Enter email address in **Email** text field. - Click the **Create Account and Continue** button.	email=EMAIL	1. Your Account page resets with Warning message—**Please enter password**. 2. **Logout** link not found.
4	**Short Password**: - Enter email address in **Email** text field. - Enter password in **Password** text field. - Enter password in **Confirm Password** text field. - Click **Create Account and Continue** button.	email=EMAIL password=SHORT_PWD confirm password=SHORT_PWD	1. Your Account page resets with Warning message—**Your password must be 8 characters or longer**. 2. **Logout** link is not found.
5	**Unconfirmed Password**: - Enter email address in **Email** text field. - Enter password in **Password** text field. - Enter password in **Confirm Password** text field. - Click **Create Account and Continue** button.	email=EMAIL password=VALID_PWD confirm password=INVALID_PWD	1. Your Account page resets with Warning message**Password does not match**. 2. **Logout** link is not found.
6	**Register Valid User**: - Enter email address in **Email** text field. - Enter password in **Password** text field. - Enter password in **Confirm Password** text field. - Click **Create Account and Continue** button.	email=EMAIL password=VALID_PWD confirm password=VALID_PWD	1. **Logout** link is found. 2. Page redirects to User Account page. 3. Message found: **You are registered as**: **e:<EMAIL>**.
7	**Click Logout link**.	User action	1. **Logout** link is NOT found.

LOGIN USER

	Test Steps	Data	Expected
1	**Click Home page**.	User action	1. **WELCOME** tab is active.
2	**Log in Wrong Password**: - Enter email in **Email** text field - Enter password at **Password** text field. - Click **Login** button.	email=EMAIL password=INVALID_PWD	1. **Logout** link is NOT found. 2. Page refreshes. 3. Warning message—**Sorry your password was incorrect** appears.

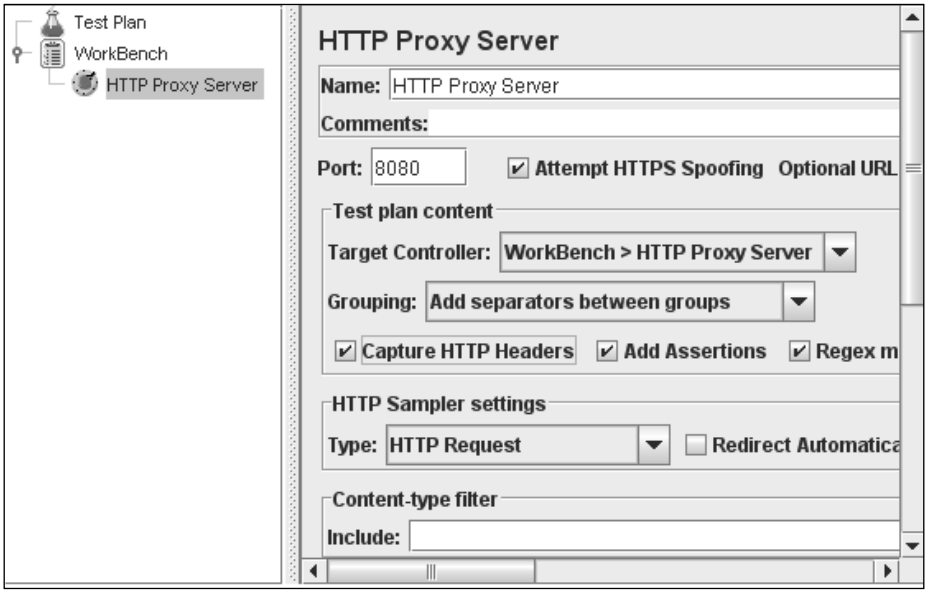
Test Steps	Data	Expected
3 **Log in Non-Exist Account**: - Enter email in **Email** text field. - Enter password in **Password** text field. - Click **Login** button.	email= INVALID_ EMAIL password= INVALID_PWD	1. **Logout** link is NOT found. 2. Page refreshes. 3. Warning message—**Sorry, this does not match any existing accounts**. **Please check your details and try again or open a new account below** appears.
4 **Log in Valid Account**: - Enter email in **Email** text field. - Enter password in Password text field. - Click **Login**-button.	email=EMAIL password= VALID_PWD	1. **Logout** link is found. 2. Page reloads. 3. Login successful message—**You are logged in as**: appears.
5 **Click Logout link**.	User action	1. **Logout** link is NOT found.

With the exception of the Configuration elements, Listeners, and Assertions, which we will add later, our **Test Plan** will take the form that you see in the following screenshot:



Using HTTP Proxy Server to Record Page Requests

As in recording requests in Chapter 5, you will need to include the HTTP Proxy Server element in the WorkBench. Some configuration will be required, as shown in the following snapshot:



Configuring the Proxy Server

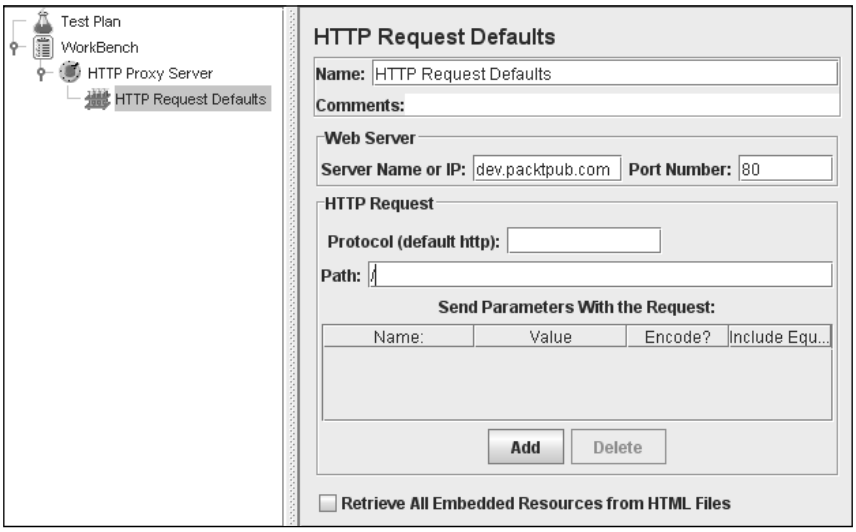
Simulating **Create Account** and **Login User** scenarios will require JMeter to make requests for the registration and login pages that are exposed via HTTPS. By default, **HTTP Proxy Server** is unable to record HTTP requests. However, we can override this by selecting (checking) the **Attempt HTTPS Spoofing** checkbox.

Selecting **Add Assertion** will be especially useful as we add specific patterns of the page that we want to evaluate as a later part of this exercise. The **Capture HTTP Headers** option is selected to capture the Header information as we begin recording. However, to make the recording neater, we will keep this option unchecked.

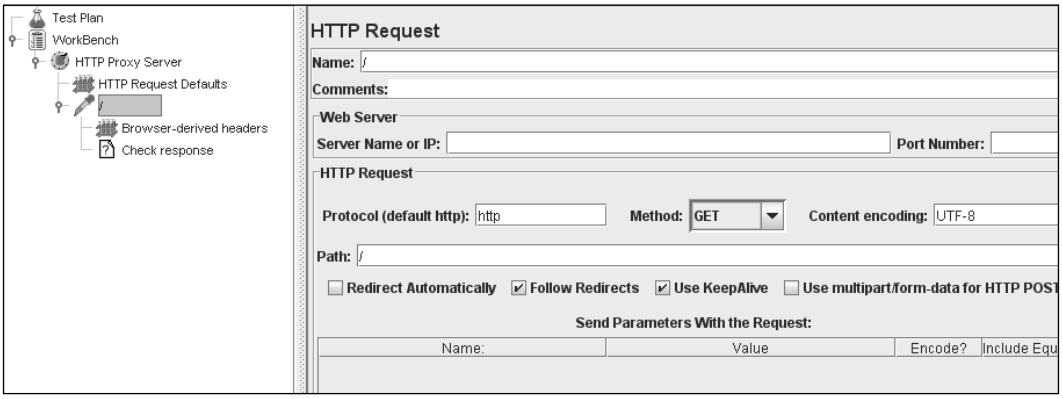
In addition, since we do not require images in our testing, in the **URL Pattern to Exclude** section, add these patterns: `.*\.jpg`, `.*\.js`, `.*\.png`, `.*\.gif'`, `.*\.ico`, `.*\.css`, otherwise these image files, which are not necessary for our testing, will be recorded causing unnecessary clutter in our recording.

Adding HTTP Request Default

A useful addition to this element is the HTTP Request Default element, a type of Configuration element. Since this Test Plan will employ multiple HTTP request elements targeting the same server and port, this element will be very useful. The web server name will not be captured for each HTTP Request sampler record, since the Request Default element will retain this information. With a little configuration change in this element, it allows the Test Plan to run even when the application is the deployed to a different server and/or port. The following snapshot is the **HTTP Request Default** element that we will use for this exercise.

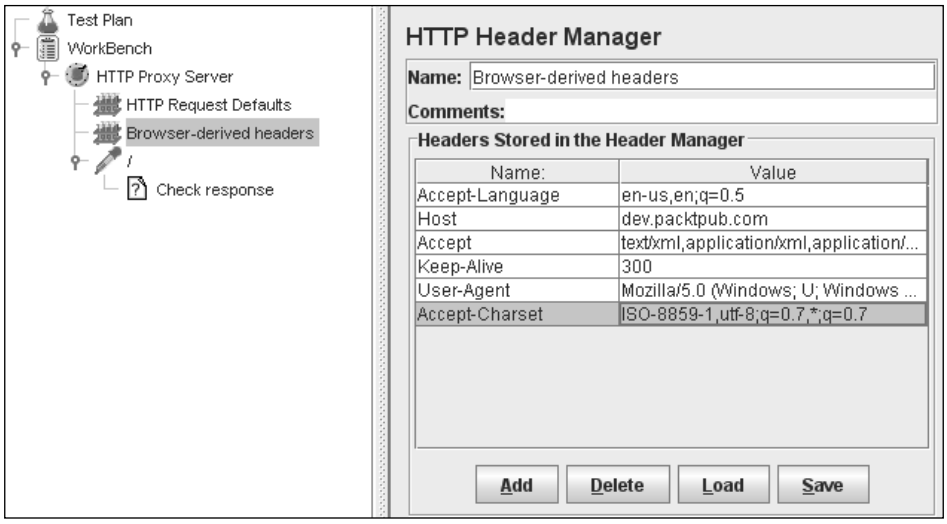


As we use this default element, our subsequent recording never needs to append the Server name. The result of our recording of the first page is shown in the following snapshot:



Adding HTTP Header Manager

Another very useful default element is the **HTTP Header Manager** Configuration element. This element can either be added to the Test Plan and configured manually as an afterthought, or we can simply use the recorded **Browser-derived headers** element as included in the recording. For convenience, we will choose the latter option. Once the Proxy Server records the homepage request, stop the recording. You will find a Header Manager for this page is being captured, as **Browser-derived header**. Simply click and drag this element to the top of the current scope of the HTTP Proxy Server. Notice that I have removed the **Referer**, since we want to create a default for the remaining HTTP Requests. Following is a snapshot of this change.

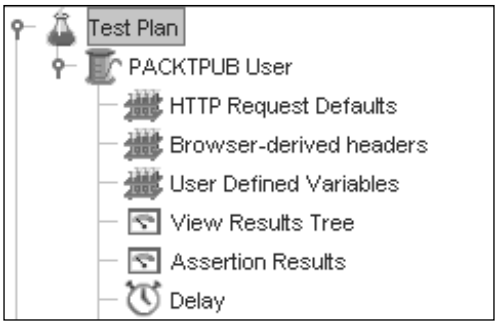


Now you may de-select the **Capture HTTP Headers** option in the Proxy Server element, since we have the default header.

Let the Recording Begin...

Let us proceed with the recording following the test cases in the previous table as our guide. As you record each page, select the specific tags or page elements the correctness of which you want to validate and add them to the **Patterns to Test** section in the **Response Assertion** element of each sampler. This may take most of your recording time, since as you record, you need to decide carefully which page element(s) would be the most effective measure of correctness. There are plenty of developer tools available to help you in this possibly tedious task. My favorite is the **Inspect Element** feature in **Firebug**, a Firefox browser add-on by Mozilla. You may choose patterns that you would expect to see or otherwise by selecting or de-selecting the **Not** option at **Pattern Matching Rules** section.

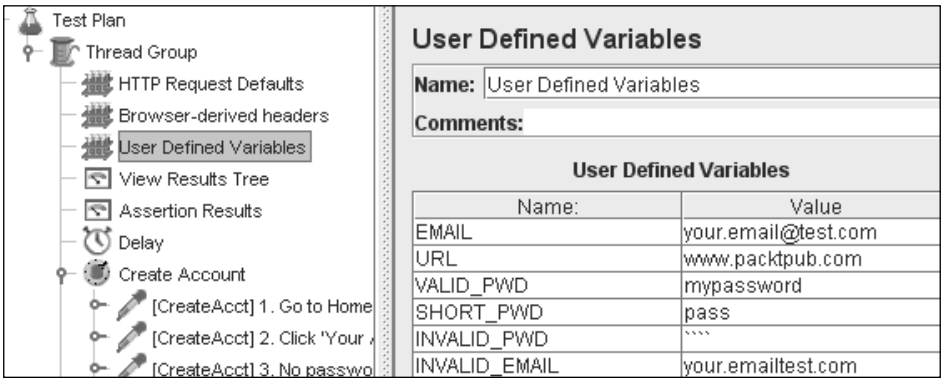
After recording is completed, you may rename and organize your samplers, as you move them to the Test Plan (refer to the following figure). You may want to add a few more Configuration elements in your Test Plan, as in my sample shown in the following snapshot:



- I have added User Defined Variables, two more Listeners, and a Constant Timer with a constant delay of 2 seconds after the request for each page was completed. The Assertion Results listener is used with the Response Assertion elements, to summarize the success or failure of a page in meeting the validation criteria defined in each Response Assertion.

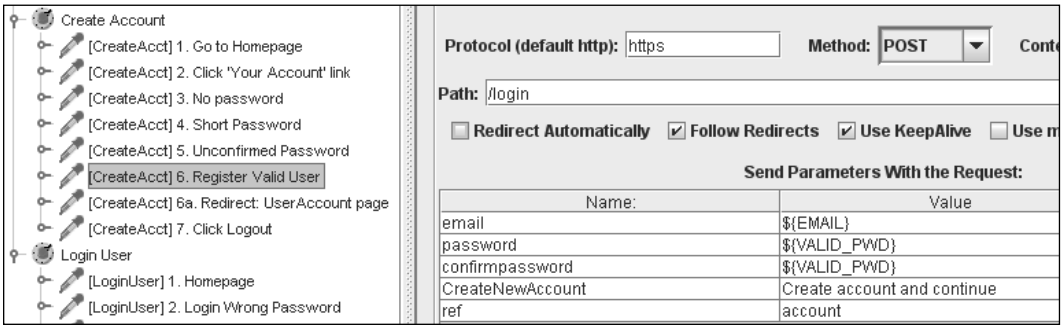
Adding User Defined Variables

The **User Defined Variables** (**UDV**) element as shown in the following snapshot is particularly interesting with regards to the test case design we drafted earlier in the table. It allows you to plug values to variables being used in various locations in the Test Plan. The JMeter Test Plan we have created will implement the exact values assigned to different variables. Following is a snapshot of the UDV I have set up for our Test Plan.

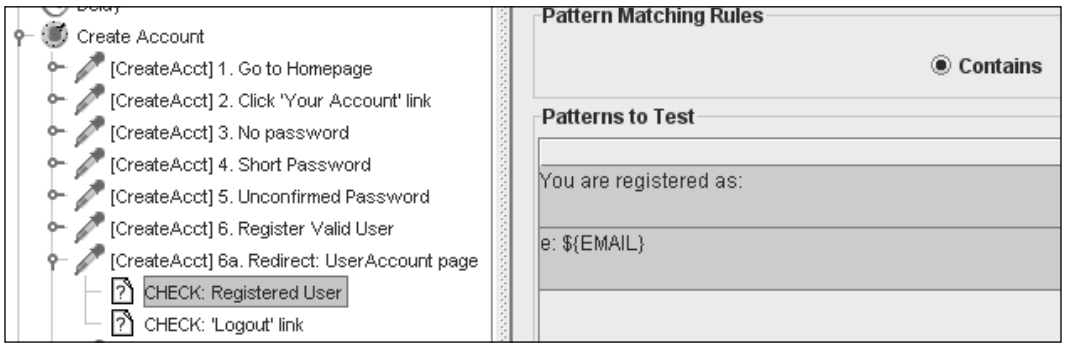


How do we use these variables in the Test Plan? Simply use the format `${Variable-name}` anywhere in the Test Plan that we want to use the value of a Variable.

For example, in the HTTP Request Sampler following **CREATE ACCOUNT | Test Step#6**: **Register Valid User**, as you can see below, the parameter **password** has value `${VALID_PWD}`, referring to the corresponding variable assigned in UDV.

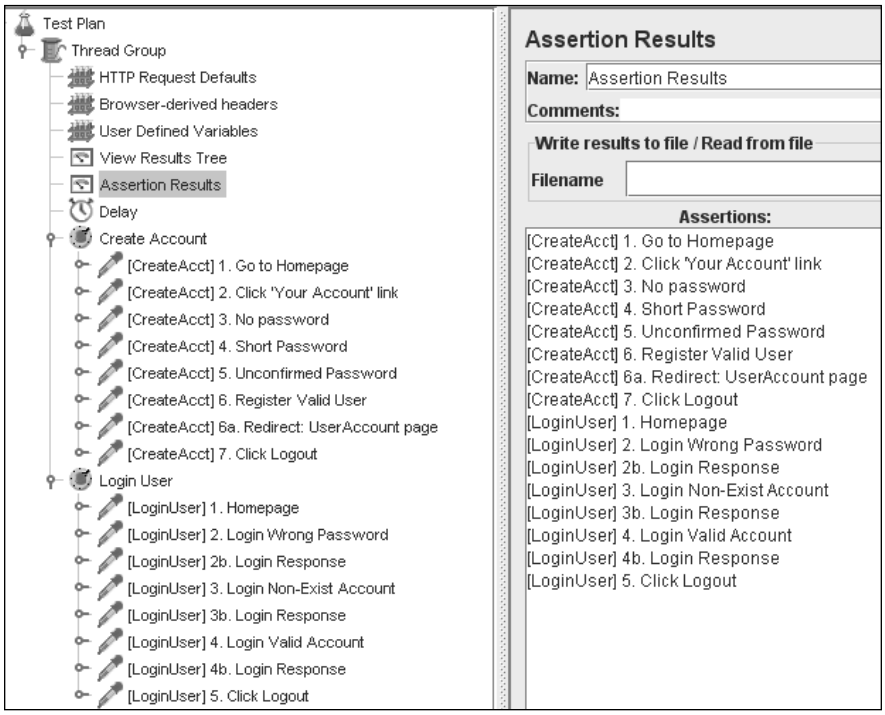


We may also use the variables set in UDV in other elements, namely Response Assertions. This feature is particularly useful when the assertion depends on varying values, such as when we want to verify URLs, verifying user names, account no, etc.—depending on the values we want to include throughout the entire testing. The following snapshot may give us a clear idea of how a UDV can be used in an Assertion element. The URL variable defined in UDV is used in the **Patterns to Test** section of this Assertion, as part of a complete page element that we want to verify in the page Sampler.

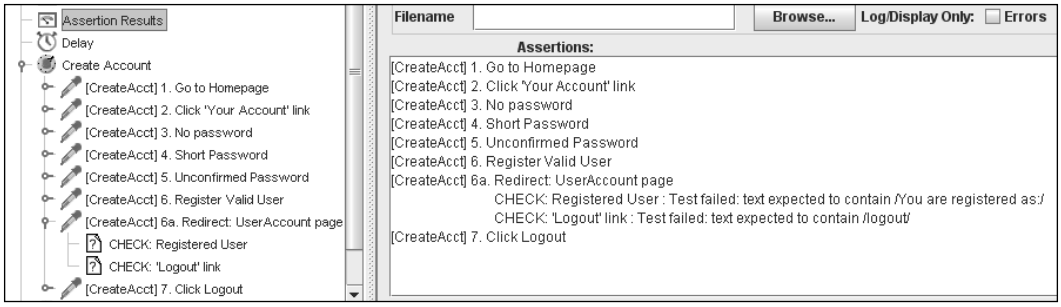


Running the Test

Once the assertions are properly completed, we are expecting that running our Test Plan would pass all the assertions. Passed assertions will not show any error in **Assertion Results | Listener** installed within the same scope. As for all Listeners, results as captured by the Listeners can be saved and reproduced at a later time. Following is a sample explaining what passed Assertions would reveal as the Test is executed.

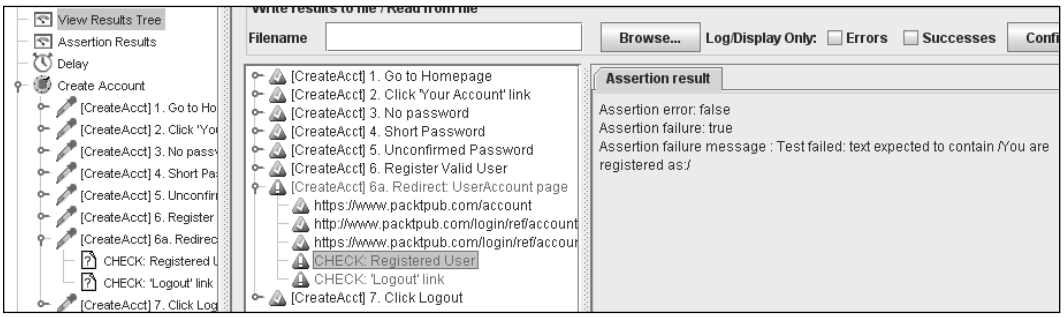


On the other hand, a failed Assertion would show an error message in the same Listener as the following snapshot illustrates.



Since a page error or **Page not found** error is a real risk in web applications, a failure may originate from such an error, and not just because of a failed Assertion. We can view more information about the sampler that contains the failed Assertion to investigate the origins of a failure. A **View Results Tree** Listener records the details of requests and logs all errors (indicated by the red warning sign and red fonts).

The following figure shows that the page was available and page request was successful, however, the assertion failed.



Summary

This chapter provided visual means for you to understand the capabilities of JMeter tools that support functional testing, as we directly wrote and implemented a JMeter script. We have demonstrated building a Test Plan to contain functional validations (or assertions) by incorporating various essential JMeter components, particularly the 'Response Assertion' element and 'Assertion Result' Listener. By using the 'User Defined Variable' Configuration element, we have also parameterized several values in order to give our Test Plan better flexibility. In addition, we have observed the result of these assertions as we performed a 'live' run of the application under test. An HTTP Request sampler may require to be modified, if there are any changes to the parameter(s) that the sampler sends with each request. Once created, a JMeter Test Plan that contains assertions can then be used and modified in subsequent Regression tests for the application. The next chapter will let us see various ways that a JMeter script can be further configured and tweaked so that it supports better portability and testability. Chapter 7 will describe various methods and tools available in JMeter that support more advanced and complex testing requirements.

Where to buy this book

You can buy Apache JMeter from the Packt Publishing website:
`http://www.packtpub.com/beginning-apache-jmeter/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com