

---

# Hello World: Rational Performance Tester

Get to the bottom of application performance issues

Skill Level: Intermediate

[Dennis Schultz \(dennis.schultz@us.ibm.com\)](mailto:dennis.schultz@us.ibm.com)  
Marketing Engineer  
IBM

12 Mar 2007

This tutorial in the [Hello World series](#) introduces you to IBM® Rational® Performance Tester and highlights its basic features. Practical, hands-on exercises teach you how to record automated performance tests, use data-driven techniques to ensure randomization, play-back tests, and evaluate real-time performance reports. Upon completing the tutorial you will be able to use Rational Performance Tester to determine the cause of performance problems in your applications.

## Section 1. Before you start

### About this series

The [Hello World series](#) is for novice developers who want a high-level, hands-on overview of IBM software products. Each tutorial in the series provides simple exercises and step-by-step instructions to familiarize you with the components and use of a particular product. Upon completing a tutorial in the *Hello World* series you will know enough about the product to begin exploring and using it on your own.

### About this tutorial

This tutorial uses hands-on exercises to familiarize you with Rational Performance Tester. Step-by-step instructions teach you how to record an automated performance test, enhance the test using built-in data-driven techniques, play-back the test as part of a performance schedule, and evaluate real-time reports to determine the root cause of a performance problem. The maximum estimated

running time for the tutorial is three hours.

## Objectives

After completing this tutorial you should understand the basic functions of Rational Performance Tester and be able to use it to discover and analyze performance problems in your applications.

## Prerequisites

This tutorial is for testers new to test automation and unfamiliar with Rational Performance Tester. As you are taking the tutorial, you can practice the steps yourself if you have access to the environment the tutorial requires. If you don't have access to the environment, you can still read the tutorial and view the animated demos. You just won't be able to try the steps for yourself.

The easiest way to access the tutorial environment is through the [Rational Performance Tester online trial system](#) created for the tutorial. The trial system uses the Citrix Access Platform to provide you with a connection from your workstation to a remote server running Rational Performance Tester, WebSphere Application Server 6.0, and the sample application to be tested. If you choose to set up the tutorial environment on your own machine, please use the [Hello World: Rational Performance Tester \(for downloadable trial code\)](#) version of this tutorial that is written for this purpose.

In order to view the animated demos for the tutorial you must enable JavaScript in your browser and install [Macromedia Flash Player 6 or higher](#).

## Animated demos

If this is your first encounter with a *developerWorks* tutorial that includes animated demos you might want to know a few things about them:

- Demos are an optional way to see the steps described in the tutorial done for you. To view an animated demo, click the given **Show me** link and the demo will open in a new browser window.
- Each demo contains a navigation bar at the bottom of the screen. Use the navigation bar to pause, exit, rewind, or fast-forward portions of the demo.
- The demos are 800 x 600 pixels. If this is the maximum resolution of your screen or if your resolution is lower than this then you will have to scroll to see some areas of the demo.
- You must have JavaScript enabled in your browser and [Macromedia Flash Player 6 or higher](#) installed to view the demos.

---

## Section 2. Getting started

### Overview of Rational Performance Tester

IBM Rational Performance Tester, hereafter known as Performance Tester, is a performance test creation, execution, and analysis tool that helps development teams validate the scalability and reliability of their Web-based applications before deployment. Many of Performance Tester's features have been explicitly designed with the novice load tester in mind. Performance Tester allows you to use one of several Web browsers (Internet Explorer, Mozilla, or Firefox) to test a Web-based application. The results of your interaction are captured and recorded on the operating system of your choice (Windows or Linux). The test is presented in a concise tree-based editor that is capable of exposing underlying details to the expert on an "as needed" basis.

Test scripts are then grouped together in various combinations to reflect the multiple types of user that comprise the projected user population. You can specify the number of simulated system users at execution time. Test execution is accompanied by easy-to-read, real-time reports that update throughout the test run. Bottlenecks based on metrics such as round-trip performance, transaction rates, and system diagnostics are highlighted in these reports.

You can also use Performance Tester to further identify the root cause of poor performance problems from the hardware- or software-component level through advanced resource monitoring and response-time tracking.

Although this tutorial focuses on testing a J2EE, Web-based application, you can use Performance Tester to test any Web-based application. You can also extend Performance Tester to test the performance of additional application types such as Siebel, SAP, and Citrix (see [Resources](#)).

### Setting up the tutorial environment

As previously mentioned, the easiest way to access this tutorial is through the [online trial system](#) developed using the Citrix Access Platform. The trial system gives you free access to Performance Tester, WebSphere Application Server 6.0, and the Adventure Builder sample application, and minimizes your installation and configuration time for the tutorial. Once you have registered for the online trial system, installed the Citrix Metaframe Presentation Server, and logged into the server, you are ready to begin. Performance Tester will launch in a Citrix client session and will appear just as if it were running on your local desktop. You can interact with it just as you would if it were installed locally. The countdown clock on the Rational Test Drive Environment Web page will keep track of the time remaining in your session.

If you choose to set up the tutorial environment locally you will need to install and configure [Rational Performance Tester](#) and [WebSphere Application Server 6.0](#) in your workstation. *Note that the Adventure Builder sample application used by this tutorial is only available with WebSphere Application Server 6.0, not 6.1.* You should also allow additional time to create and configure a project in which to store your test artifacts before starting.

The tutorial is written from the assumption that you are using the online trial system.

### Performance Tester and Eclipse

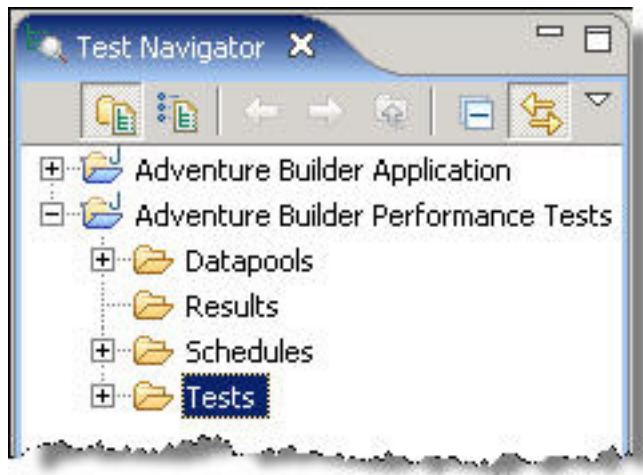
Performance Tester is based on the open-source Eclipse platform. Eclipse provides a rich working environment for many tools, both open source and commercial, including many of the offerings of the Rational Software Delivery Platform such as Rational Software Architect, Rational Application Developer for WebSphere Software, and Rational Functional Tester (see [Resources](#)). This provides a common user experience for tooling across the software development life cycle. Not only are these tools all based on Eclipse, but often times they actually share the same shell. In other words, the capability of each of these tools is presented to the user as a new perspective in the same shell. A *perspective* is a consolidation of tools and views focused on one particular task. The perspective for Performance Tester is known as the *Test perspective*. As the name implies, the Test perspective provides views that are needed by a developer or QA professional focused on testing a software application or system.

## The tutorial workspace and sample projects

Assuming you are using the online trial system, Performance Tester will be associated to a pre-configured workspace. A *workspace* can be any directory location where your work is stored. In your case, this workspace contains two projects. The first project is the **Adventure Builder Application**. This project contains the source code for the Adventure Builder sample application you will be testing. The application has already been deployed to WebSphere Application Server on the online trial system. The project is only in your workspace so that Performance Tester can navigate to the source code later when you are attempting to find the root cause of a performance bottleneck.

The second project, **Adventure Builder Performance Tests**, is the test project you will use to store your tests, datapools, schedules, and results. If you expand the project, you will see several folders used to organize your test assets. You can add, remove, and customize folders as you like. There isn't much to examine in this project yet. You will look more closely at it once you have recorded a test.

### Figure 1. A Performance Tester workspace viewed in the Test Navigator



## Section 3. Record a test scenario


Performance tests are most often created by recording your manual interactions with the system under test. In this section of the tutorial, you will use the automated HTTP recorder to capture the scenario of interacting with the Adventure Builder sample application to construct a vacation package and purchase it.

The Adventure Builder application is already running in the online trial environment. All you need to do is connect to it through a Web browser while the recorder is engaged.

### Browser support

Although you are using Internet Explorer in this tutorial, Performance Tester can work with any Web browser that supports SOCKS proxies. Performance Tester can automatically launch Internet Explorer, Mozilla, and Firefox.

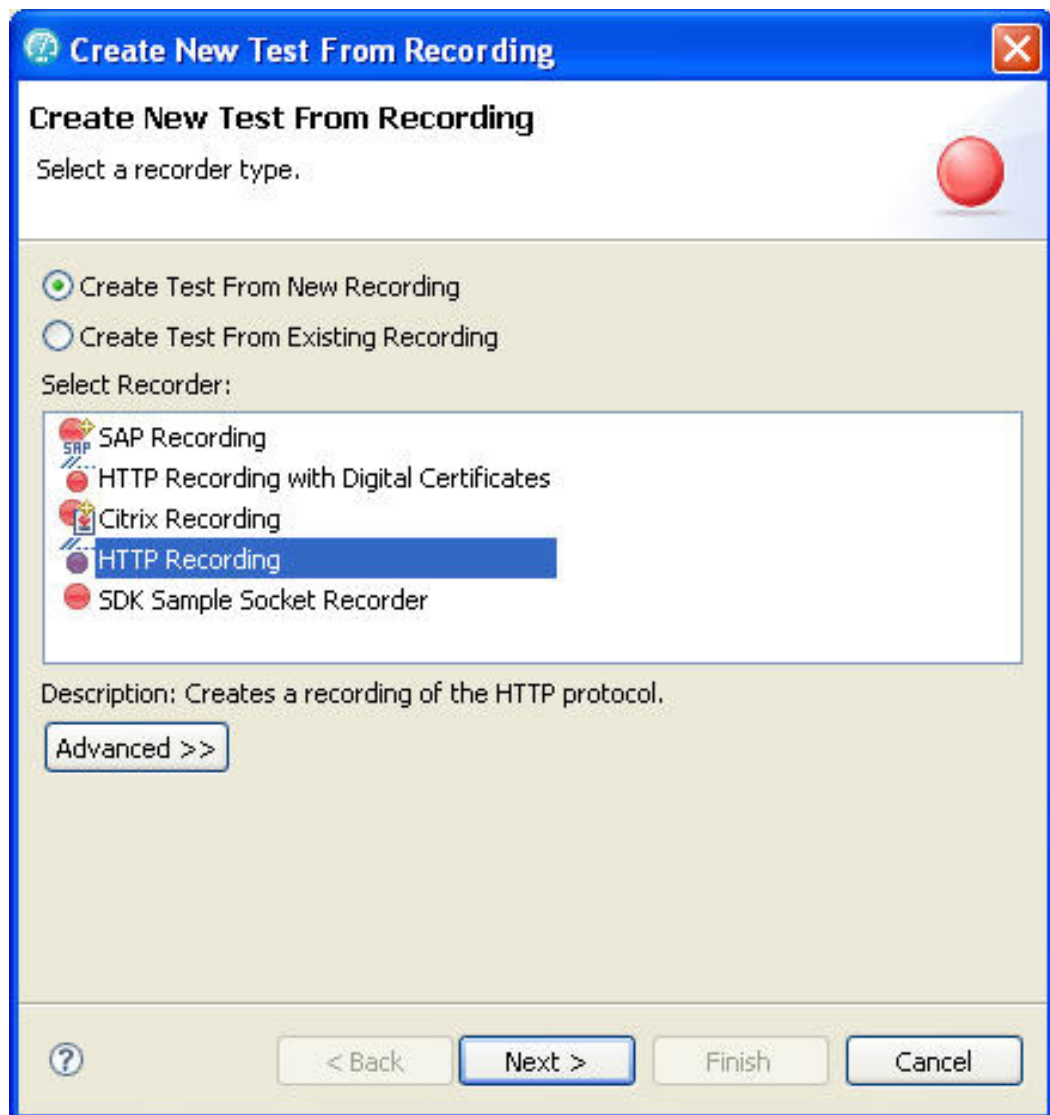
### Animated demo

Would you like to see these steps demonstrated for you?  [Show me](#)

## Start the recorder

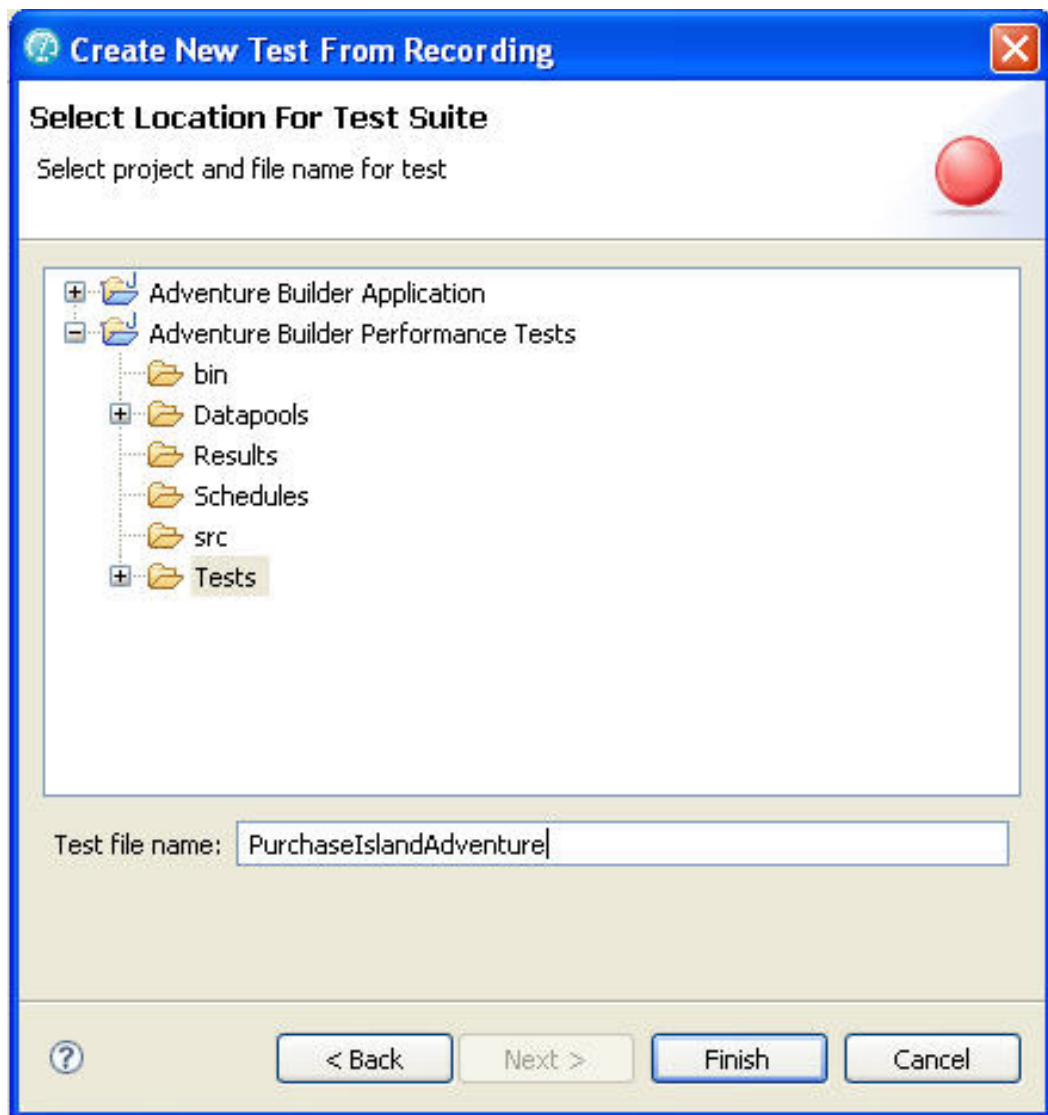
1. Start the recorder by clicking **Create a Test from Recording** on the toolbar . This opens the Create New Test From Recording window.

**Figure 2. The Create New Test From Recording window**



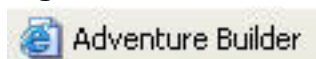
2. Select **HTTP Recording** and click **Next**.
3. On the next page of the wizard, select **Adventure Builder Performance Tests > Tests** as the location to create the file.
4. Enter `PurchaseIslandAdventure` as the test-file name and click **Finish**.

**Figure 3. The Create New Test From Recording window, page 2**



5. The recorder is engaged and Internet Explorer is launched to the "Welcome to Performance Testing" page. Clear the cache of temporary files by selecting **Internet Explorer Tools > Internet Options ....** Under Temporary Internet Files, click **Delete Cookies** and confirm. Then click **Delete Files**. Check **Delete all offline content** and click **OK** to confirm.
6. Click **OK** to dismiss the Internet Options window.
7. Launch the Adventure Builder application by clicking the Adventure Builder button in the Internet Explorer Links toolbar.


**Figure 4. Launch the Adventure Builder application**



## Navigate the application



**Animated demo**

Would you like to see these steps demonstrated for you?  [Show](#)  
[me](#)

Now that the recorder is engaged, you will navigate the application just as you normally would.

1. In the left navigation area of the page, click **Island Adventures**.  
**Figure 5. Island Adventures link**



2. Here you see an expanded list of island adventure trips. From the submenu on the Available Adventure Packages page, click **Maui Survival Adventure**.

**Figure 6. Maui Survival Adventure link**

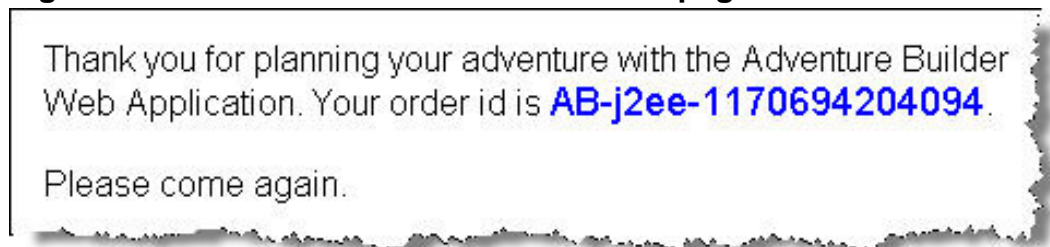


3. This page shows you the options available for the package you have chosen. Click **Select Package** on the right side of the Adventure Package Details page.



4. You are taken to the Package Options page. Here you can personalize your vacation package by changing the number of people, start date, number of days, etc. For now, just accept the defaults by clicking **Set Package Options**.
5. On the Adventure Package Details page, click **I Will Provide My Own Transportation**.
6. On the Adventure Package page, click **Checkout**.
7. Let's keep it simple for now: on the Sign On page, click **Sign In** as a returning customer.
8. On the Enter Order Details page, scroll to the bottom and click **Submit**.
9. After a moment, you will be taken to the Checkout page. Here you should see your order ID.

**Figure 7. The Order ID link on the Checkout page**



Note that your order ID will be unique and will not be exactly as shown here. Wait for several seconds after the checkout page loads, then check on the status of your order by clicking on the **Order ID** link.

10. Once the Order Tracking Results page loads, *close the browser*. This will cause Performance Tester to begin generating the test based on the traffic it has captured.

---


## Section 4. Review and customize the test

Performance Tester generates a test based on the HTTP traffic it captured during the recording. The test is much more than a simple HTTP trace log, however. Behind the scenes, Performance Tester does a lot of processing to create a test that is robust, extensible, and easy to maintain. In this section, you will examine the generated test in greater detail and customize it to use unique data.

### Examine the test

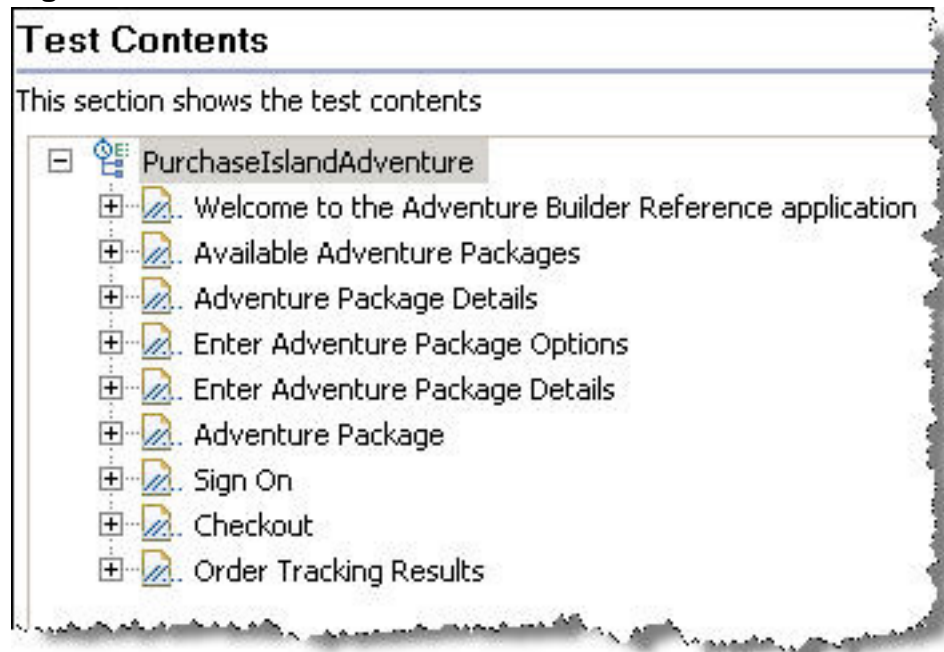
---

**Animated demo**

Would you like to see these steps demonstrated for you?  [Show](#)  
[me](#)

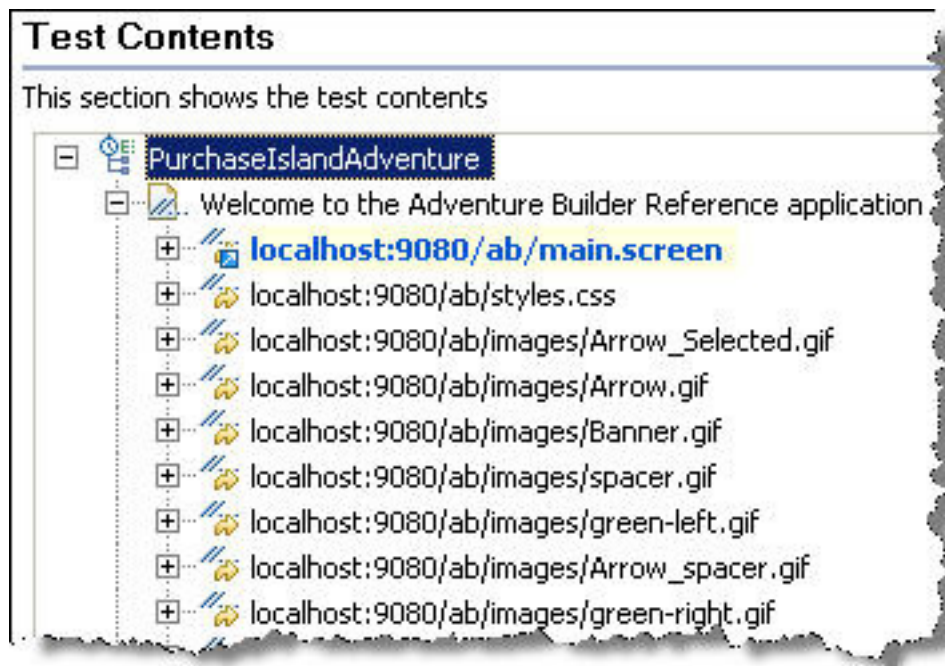
The test is represented in a tree format in the left portion of the Test Editor view. Each top-level node in the tree represents a Web page visited during your recording session. The name of the node is based on the name of the Web page.

**Figure 8. The Test Contents tree view**



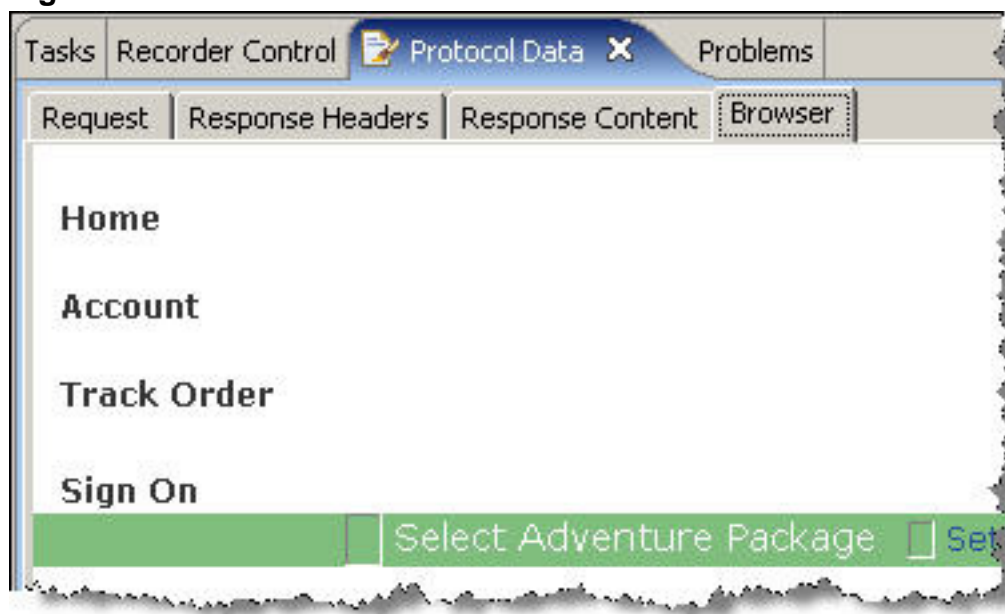
1. Expand the **Welcome to the Adventure Builder Reference application** node. Here is where the advanced performance test engineer can see all the details of the transactions behind the page. The first element is highlighted in blue to indicate that it is the primary request -- the request for the page HTML contents.

**Figure 9. The expanded page in the Test Contents tree view**



2. Click on the **Protocol Data** view in the bottom portion of the window, then click on the primary request highlighted in blue in the tree. The details of the request and its corresponding response are shown in the Request, Response Headers, and Response Content tabs of the Protocol Data view. The Browser tab even renders the contents of the selected element.


**Figure 10. The Browser tab in the Protocol Data view**



3. Detailed information about the selected element is also presented in the right-hand portion of the Test Editor view. You can edit this data if you need to change the host, URL, request header values, or any other field.

## Automatic data correlation

**Animated demo**

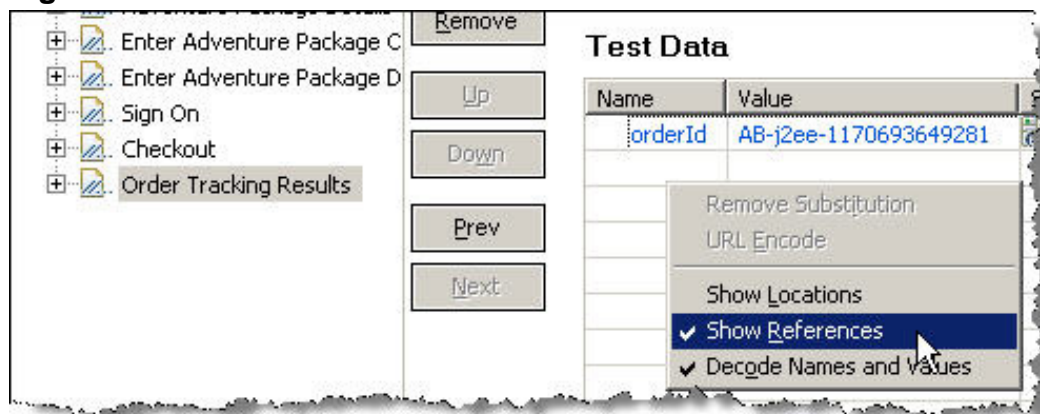
Would you like to see these steps demonstrated for you?  [Show me](#)

Web applications tend to be highly dynamic. For example, in the scenario you recorded you placed an order for a vacation package and were given a unique order ID. You then used that order ID to check the status of your purchase. When you play-back this test, it will place another order and you will be given a different order ID. You would want Performance Tester to check the status of that new order ID, not the one you previously recorded.

For that reason, Performance Tester performs automatic data correlation. That is, it looks at data parameters sent to the server and matches them up with prior response data from the server. Accessing the correlated data is easy.

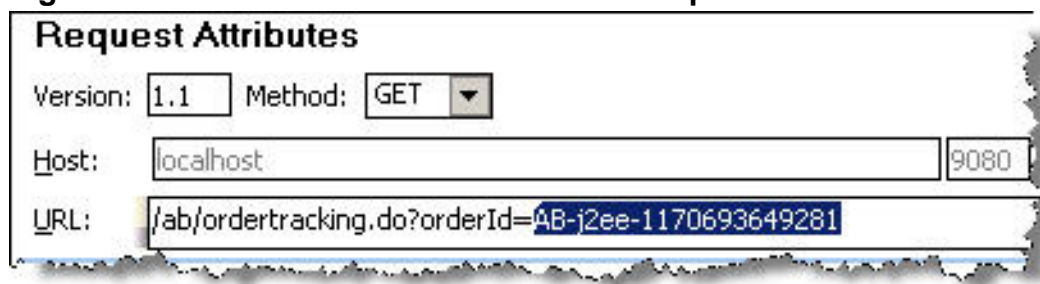
1. Highlight the **Order Tracking Results** node in the Test Contents area of the Performance Test view. Now right-click inside the Test Data area to the right and select **Show References**. Notice that the **orderId** field is being substituted with data from a prior response.

**Figure 11. Show References for data correlation**



2. Double-click **orderId**. This takes you to the URL of the actual request for that page.

**Figure 12. Correlated data in the URL of a request**



3. Right-click the highlighted string and select **Go To**. This takes you directly to the **orderId** value in the response text of the Checkout request. During test playback, Performance Tester will substitute the orderId value it receives in this response for the orderId in the request for order tracking.

**Figure 13. Correlated data in the prior response**

Granted, you now know far more than you probably wanted to know about data correlation; but that is the beauty of Performance Tester: it does all this for you without any hand coding or other effort on your part.

## Randomize the data

### Animated demo

Would you like to see these steps demonstrated for you? [Show](#)

[me](#)

In performance testing it is essential to be able to randomize the data being sent to the server. Modern Web applications have many layers of caching. If you were to emulate a thousand users using the application doing exactly the same thing, you would not observe typical performance behavior. Once the first emulated user performed the transaction, all the subsequent users would be drawing information from the cache. For this reason, performance test engineers often spend much of their time configuring tests to pull random data from a "datapool" so that each emulated user uses unique information.

Performance Tester automatically identifies likely candidates for datapool access and makes it possible to associate these fields with data sources you provide.

1. Select the **Enter Adventure Package Details** page node in the test contents. Notice in the Test Data area the `start_month`, `start_year`, and `start_day` parameters. These were the default values in the Options page, which were subsequently transmitted back to the server when the Set Package Options button was clicked.

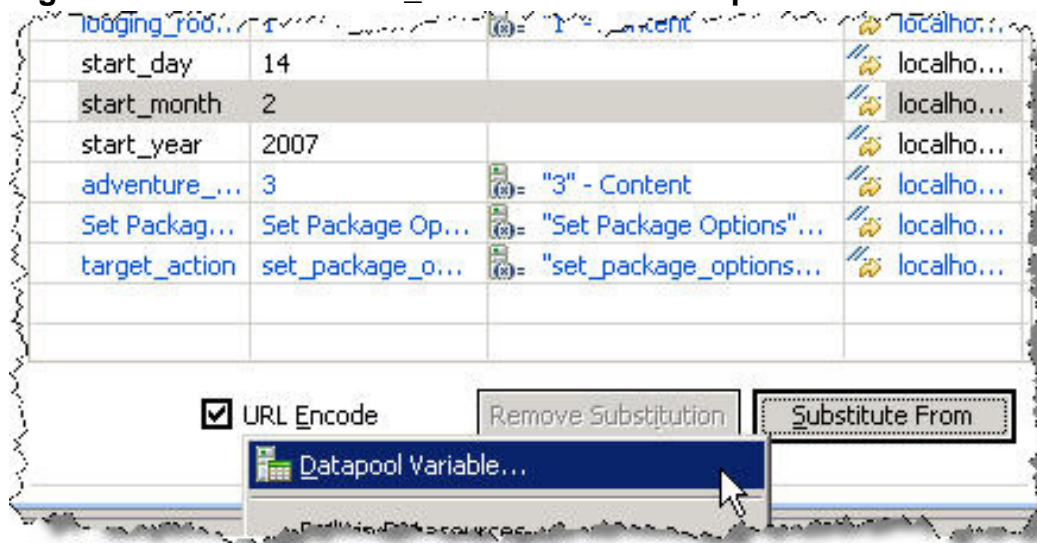
**Figure 14. Datapool candidates**

2. Set up a datapool to randomize the values used by your virtual users when you play-back this test. Select **start\_month** in the Test Data area.



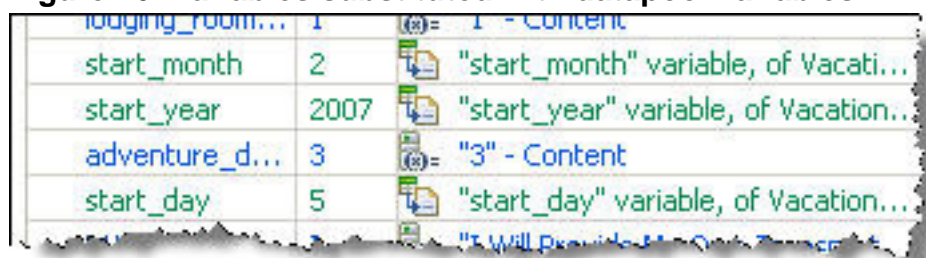
3. Click **Substitute From** below the Test Data area.
4. Select **Datapool Value....**

**Figure 15. Substitute start\_month from the datapool variable**



5. Click **Add Datapool** on the "Select datapool" column window. You will add an association to an existing datapool to this test.
6. Select the existing **VacationStartDates** datapool and click **Select**.
7. Back in the "Select datapool" column window, select **start\_month** and click **Use Column**. Note that the "Substituted with" column next to the start\_month variable now has a reference to the datapool column.
8. Select **start\_year** in the Test Data area. Repeat the process. This time you will not need to add the datapool reference; just select start\_year and click **Use Column**.
9. Repeat the above procedure for **start\_day**. The three variable rows should be highlighted in green to indicate they are being substituted from the datapool and should show references in the "Substituted with" column.

**Figure 16. Variables substituted with datapool variables**



10. Press **Ctrl-S** to save the test when finished.

## Section 5. Schedule a workload

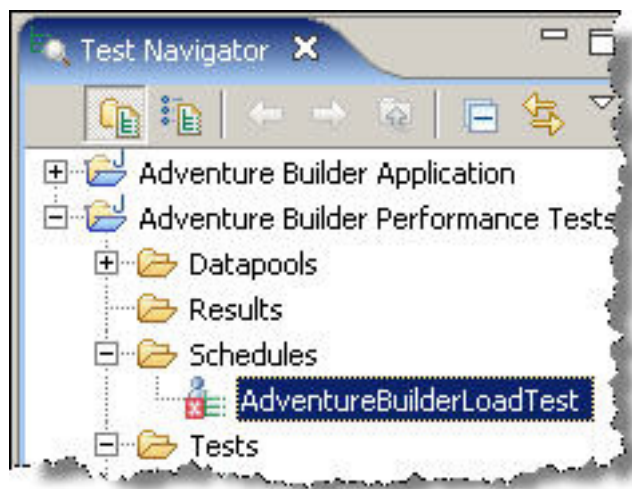
One of the keys to a successful performance test is the ability to accurately model the anticipated system workload. Software systems typically have various types of users that perform varied tasks. Performance Tester provides a graphical interface to enable you to model your user activities.

### Create a schedule

#### Animated demo

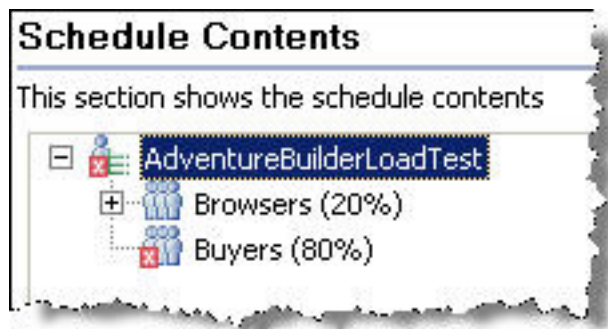
Would you like to see these steps demonstrated for you? [Show me](#)

1. Expand the Schedules folder and double-click the **AdventureBuilderLoadTest** schedule to open it in the Schedule Editor view. This is a schedule that has been partially completed for you.  
**Figure 17. AdventureBuilderLoadTest schedule in Test Navigator view**



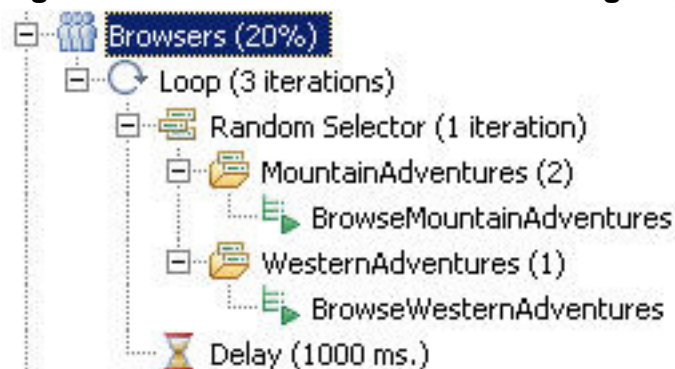
2. Two user groups have already been defined in this schedule. As you can see from the annotations on the Schedule Contents, browsers represent 20 percent of your emulated users while buyers represent 80 percent.  
**Figure 18. Schedule Contents shows defined user groups**





3. Expand the **Browsers** user group and all contained elements in the Schedule Contents.

**Figure 19. Details of the Browsers user group**



Performance Tester offers many advanced constructs to help you test the impact of a realistic load on your system. The hierarchy under the Browsers user group can be interpreted as follows: Each browser will peruse three adventure packages. Browsers will randomly choose to look at either mountain adventures or western adventures but are twice as likely to look at mountain adventures. The action of browsing the adventure category is emulated by tests that were recorded in much the same way you recorded your test. After looking at each category, a browser will wait 1000 milliseconds before browsing another category.


4. The Buyers group has not been completed. Use the test you recorded earlier as an implementation for the Buyers test. Click the **Buyers** user group.
5. Click **Add > Test** and select **PurchaseIslandAdventure** from the Select Performance Tests window. Save the AdventureBuilderLoadTest schedule.

## Section 6. Run an automated performance test

In this section you will learn how to execute your test against the Adventure Builder application and monitor the results.

## Launch the test

### Animated demo

Would you like to see these steps demonstrated for you?  [Show](#)


[me](#)

1. Select the topmost node in the Schedule Contents -- that is, the **AdventureBuilderLoadTest** node. A number of execution options are available in the tabs on the right-hand portion of the Performance Schedule Editor view. You can specify the number of users you want to emulate in the **Number of users** field on the General tab. Since the user groups in this schedule have been defined in terms of percentages, Performance Tester will do all the adjustments for you each time you change the size of your schedule. Leave the Number of users set to 5. The online trial environment is configured for a maximum of five emulated users and attempting to run with more users will result in a license error.
2. Click **Run** on the toolbar. This launches your performance test.  
**Figure 20. Run button on the toolbar**



## Monitor the test

### Animated demo

Would you like to see these steps demonstrated for you?  [Show](#)

[me](#)

While your test is running, you can monitor its progress in near real-time. The Overall tab of the Performance Report view will show you test progress in the bar across the top. The bar graph will show the status code success rate for pages and elements. Both should show 100%. While the test is running you can browse the various tabs on the report to see what is happening.

1. Select the Summary tab along the bottom of the report. Basic statistics about the test run, pages, and page elements are given here. Note that these statistics will continue to update until the run has completed.
2. Select the Page Performance tab. This tab presents a bar graph of the average page response time for the 10 pages with the highest times.
3. Feel free to browse through the other report pages. The default information presented is relatively easy to read and gives you quick insights into the performance of Adventure Builder application. Note that the graph on the Resources tab will be blank: this is expected.

4. When the test has finished (note the progress bar on the Overall tab), go back to the Page Performance tab. You should notice that the Checkout page is considerably slower than any of the other pages.


---

## Section 7. Analyze the root cause

At this point, you have successfully used Performance Tester to uncover a performance problem in your application. The next question you will ask yourself is, "What is causing the problem?" To get to the bottom of this question you will use Performance Tester's Root Cause Analysis facilities. In this section, you will re-run your test with additional data collection tools engaged. The additional information will help you determine if you are facing a hardware or software issue and drill down to the root cause of the performance bottleneck.

### Engage resource monitoring

#### Animated demo

Would you like to see these steps demonstrated for you?  [Show](#)  
[me](#)

1. Double-click the **AdventureBuilderLoadTest** schedule in the Test Navigator view.
2. Revisit some of the additional execution controls on the schedule: start by selecting the top node of the Schedule Contents again.
3. Select the Resource Monitoring tab in the Schedule Element Details area. Resource monitoring enables Performance Tester to log any system parameter from Windows `perfmon`, Unix or Linux `rstatd`, or Tivoli Monitoring.
4. Check **Enable resource monitoring**.
5. Click **Add New...** to define a new server on which to monitor resources. Note that you may need to scroll down the right portion of the Performance Schedule view to see the button.
6. In the Resource Monitoring window, enter `localhost` as the host name. Check **Windows Performance Monitor**. Note that in this trial environment, the Web server, application server, and Performance Tester system are all running on a single machine: `localhost`. This is, of course, not a realistic situation. In a true performance testing environment, you

can define any machines that might be part of your application or test system.

7. Select the Resources tab. After a few seconds, you can see the extensive list of counters available. To keep it simple, deselect all counters except **Memory > Pages/sec** and **Processor > % Processor Time**.
8. Click **OK** to close the Resource Monitoring window.

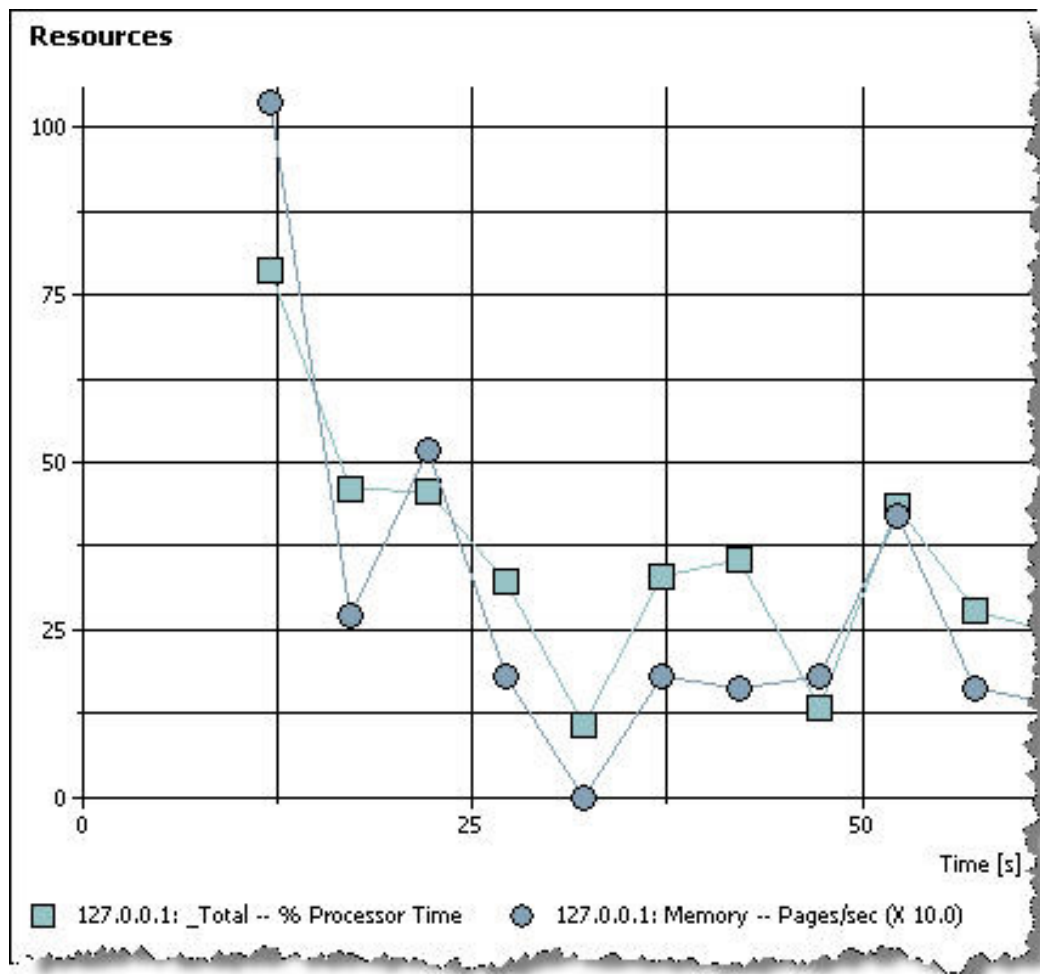
## Engage Response Time Breakdown

1. Select the Response Time Breakdown tab in the Schedule Element Details area of the Performance Schedule.
2. Check **Enable collection of response time data**. This enables Performance Tester's response-time data collection infrastructure.
3. Since you know the only test that actually visits the Checkout page is PurchaselslandAdventure, select only that test.
4. In the Options area, set the Detail level to **High**.
5. Save the schedule by pressing **Ctrl-S**.
6. Click the Run button on the toolbar again. Performance Tester launches the test just as before, but this time with resource monitoring and response-time breakdown collection engaged.

## Examine resource-utilization data

1. While the performance schedule is executing, select the Resources tab on the Performance Report. This time, you'll see data for the resources you chose to collect in the schedule.

**Figure 21. The Resources tab on the Performance Report**



2. The data you see here, although accurate, is not really representative of a typical load test. In this trial environment, the Performance Tester load generation, Web server, application server, and database server are all running on a single machine. You would normally track resources on each tier of your application. In addition, the load test you just ran was of very short duration. Normally, performance tests will be significantly longer, allowing the systems to reach a steady state. Nonetheless, the trial system gives you a sense of how easy it is to track resource utilization during a performance test.

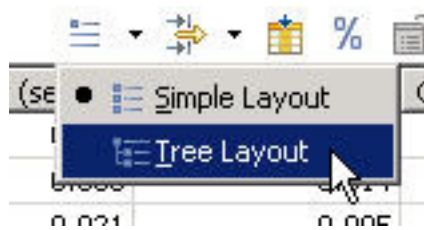
If you have a concern, you can now go to the Response vs. Time Detail tab, right-click on the graph, and click **Add/Remove Performance Counters** to overlay resource counters onto page response data. This helps you visually correlate any spikes in resource utilization with page activity.

3. Since it doesn't appear that the performance issue with Adventure Builder is hardware related, use the response-time breakdown data to find out if it is software related.

## Examine the response-time breakdown data

1. Select the Page Performance tab of the Performance Report.
2. Drill down into what went on behind the scenes for the Checkout page. Right-click on the bar in the graph for the Checkout page and choose **Display Response Time Breakdown Statistics....**
3. Select the **/ab/checkout.do** URL from the Selection Wizard and click **Finish**. The Response Time Breakdown Statistics view lists methods called on the server tiers of the Adventure Builder application. There are various ways to examine this information.
4. Switch to the Tree Layout view using the Layout button in the upper-right corner of the view.

**Figure 22. The Tree Layout view**



Now sort by *descending cumulative time* by clicking twice on the **Cumulative Time** column header.

## Find the root cause of the problem

The top node labeled **rationaltd** represents the machine. In this trial, all system-under-test and test-harness components are running on a single machine. In a real-world test, you would likely see multiple machines listed.

The second-level node labeled **J2EE/WebSphere...** is the WebSphere application server component. From the information here, you can quickly see that the J2EE facet type consuming the most cumulative time is the Servlet.

1. Expand the **Servlet** node and the **com.sun.j2ee.blueprints.waf.controller.web** package, and the **MainServlet** class nodes. This tells you that the four invocations of the **doPost** method in the **MainServlet** class consumed 42.113 seconds. Note that your actual values will probably differ.

**Figure 23. Response time of the doPost method of MainServlet**



**Page Performance > Response Time Breakdown Statistics**

localhost:9080/ab/checkout.do

Component	Base Time...	Averag...	Cumulative ...	Calls
rationaltd	167.055	41.651	294.216	196
J2EE/WebSphere/6.0.0.1/rationaltd	82.632	20.546	209.793	184
Servlet	0.379	0.095	43.105	8
com.sun.j2ee.blueprints.w	0.306	0.076	42.113	4
MainServlet	0.306	0.076	42.113	4
doPost(javax.serv	0.306	0.076	42.113	4

- Right-click the `doPost` method and choose **Open Source**. Well, what do you know -- you have located the source of your performance problem!

**Figure 24. A sleep statement in the source code**

```
public void doPost(HttpServletRequest request, HttpServletResponse
    throws IOException, ServletException {
    try
    {
        // Congrats - You found the needle in the haystack
        System.out.println("This code is executing a 10 second sleep");
        Thread.sleep( 10000 );
    }
    catch ( InterruptedException ie ) {}
}
```

## Section 8. Summary

This tutorial has introduced you to IBM Rational Performance Tester. In a very short time you were able to construct a test suite by recording a performance test for a Web application, customizing the test to randomize data upon playback, and using that test as part of a realistic performance test schedule. You executed that schedule on a small scale and used the near real-time reports to identify a slow page. Once you had identified the page, you gathered resource utilization data and response-time breakdown statistics, which you then used to investigate possible hardware and software causes. You then drilled down to the specific source code method causing the problem.

While you may have learned a lot in this tutorial, you have only scratched the surface of what Performance Tester can do. As you continue to explore Performance Tester you will discover many more features to assist you in testing the performance of your applications and releasing them with confidence. See [Resources](#) to learn more about Performance Tester and other IBM software products covered in the [Hello World tutorial series](#).



# Resources

## Learn

- The [Hello World: Rational Performance Tester \(for downloadable trial code\)](#) is another version of this tutorial that is written for people who would prefer to set up their own environment, rather than use the online trial of Rational Performance Tester.
- The [Hello World series](#) of hands-on tutorials introduces IBM software products that play a critical role in implementing an SOA foundation in your enterprise.
- See the [Rational Performance Tester](#) product page for technical documentation, how-to articles, education, downloads, and product information about Rational Performance Tester.
- "[Using IBM Rational Performance Tester to find bottlenecks](#)" (David M. Chadwick, developerWorks, September 2006) presents a real-world case study in using Rational Performance Tester.
- [IBM Rational performance testing solutions](#) provide scalability and load testing for J2EE, Web-based, Siebel, Citrix, and SAP applications.
- Learn more about the [IBM Rational Software Delivery Platform](#) -- a complete set of tools to build, integrate, modernize, extend, and deploy software and software-based systems.
- Learn more about the [Eclipse development environment](#).

## Get products and technologies

- Download a free trial version of [Rational Performance Tester](#).
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

## Discuss

- [Participate in the discussion forum for this content.](#)

## About the author

Dennis Schultz

Dennis Schultz joined Rational in 1995 as a technical sales engineer. For eight years, he worked closely with numerous clients, implementing Rational solutions in their projects. Dennis helped deploy solutions for software configuration management, change management, requirements management, and test management and implementation. Since 2003, Dennis has been a Technical Marketing Engineer for IBM Rational software. Dennis holds a B.S. in computer engineering from Iowa State University. He is based in St. Louis, Missouri, and fills his non-work time with his four children.