Hello World: IBM Rational Tester for SOA Quality and IBM Rational Performance Tester Extension for SOA Quality

Testing Web services to ensure high-quality service-oriented architecture

Skill Level: Intermediate

Brian Bryson
Technology Evangelist
IBM

15 May 2007

Get a high-level overview of both IBM® Rational® Tester for SOA Quality and IBM® Rational® Performance Tester Extension for SOA Quality. By following the step-by-step instructions, you get hands-on experience in creating a service-oriented architecture (SOA) Web service test, enhancing the test through the built-in data-driven techniques, and executing the test. This tutorial begins with the functional testing capabilities of Rational Tester for SOA Quality and concludes with an example of performance testing using the Rational Performance Tester Extension for SOA Quality.

Section 1. Before you start

About this series

This Hello World tutorial series is for novices who want high-level overviews of IBM software products. The modules are designed to introduce the product and help you decide whether you want to explore it further. The exercises cover only the basic concepts, but they are enough to get you started.
About this tutorial

This tutorial provides a high-level overview of both IBM® Rational® Tester for SOA Quality and IBM® Rational® Performance Tester Extension for SOA Quality. By following the step-by-step instructions, you get hands-on experience in creating a service-oriented architecture (SOA) Web service test, enhancing the test through the built-in data-driven techniques, and executing the test. This tutorial begins with the functional testing capabilities of Rational Tester for SOA Quality and concludes with an example of performance testing using the Rational Performance Tester Extension for SOA Quality.

Here are the major steps that you will complete:

1. Record a test which uses a temperature conversion Web service to convert temperatures between Fahrenheit and Celsius
2. Edit the test so that it will run with multiple datasets and validate the returned data from the Web service
3. Create a schedule for test execution which will run multiple iterations of the test
4. Execute the test and examine the test log
5. Optionally, rerun the test with multiple users and examine performance reports

About the application under test here

For this tutorial, you'll use a live Web service that was active as of the time of writing, April 2007. The Web service converts temperatures from Celsius to Fahrenheit. Although simple, it serves well to illustrate all of the key concepts of SOA application testing.

Objectives

After completing this tutorial, you will have gained the knowledge necessary to perform both functional and performance testing of SOA Web services without GUIs.

Prerequisites
This tutorial is for testers new to testing SOA Web services. No prerequisite knowledge of SOA applications, Web services, or the tools is assumed or required.

To complete the exercises in this tutorial, install IBM Rational Tester for SOA Quality. If you want to run the performance testing example, you also need to install the Rational Performance Tester Extension for SOA Quality.

The exercises in this tutorial are use a live Web service. Therefore, you need Internet access to complete all of the exercises.

To view the demos included in this tutorial, JavaScript™ must be enabled in your browser and Macromedia Flash Player 6 or higher must be installed. You can download the latest Flash Player on the Macromedia site.

About animated demos

If this is your first encounter with a developerWorks tutorial that includes demos (demonstrations), here are a few things that you might want to know:

- Demos are an optional way to see the same steps described in the tutorial. To see an animated demo, click the Show me link. The demo opens 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, you will need to scroll to see some areas of the demo.
- JavaScript must be enabled in your browser and Macromedia Flash Player 6 or higher must be installed.

Section 2. Difference between Rational SOA testing tools

Web services are one of the fundamental elements of a service-oriented architecture (SOA) application. Web services offer application architects and developers a way to offer a public, programmatic interface to the underlying functions of their applications. For simplicity, you can think of a Web service as a Web-accessible or Web-enabled API.
Any given application may use either its own Web services or the Web services offered by any external system. Web services are quickly becoming the building blocks of modern applications. As this trend continues, it becomes increasingly critical to ensure their proper functioning of individual Web services before incorporating them blindly into a larger system. This is the purpose of Rational Tester for SOA Quality and of Rational Performance Tester Extension for SOA Quality.

Rational Tester for SOA Quality

Rational Tester for SOA Quality is a functional regression testing tool for Web services. With this tool, you will be able to create tests to validate the proper functioning of a single Web service or a group of Web services. Unique to this tool is its ability to enable users to create these tests without having to program any special code. By using a unique, code-free design, the tool enables testers of all levels of experience to test the simplest to the most complex Web services.

Rational Performance Tester Extension for SOA Quality

Rational Performance Tester Extension for SOA Quality is a performance testing tool for Web services. By using the same architecture as Rational Tester for SOA Quality, the Rational Performance Tester extension enables testers to simulate a large volume of simultaneous calls to a Web service. With this tool, testers can assure themselves that their Web services can withstand any given transaction volume, from hundreds to hundreds of thousands.

Section 3. Tutorial setup

This tutorial assumes that you are starting with a fresh installation of Rational Tester for SOA Quality. Before you can begin your testing, you'll need to perform these two setup tasks:

- Create a project to store your tests and results
- Import the Web service definition information into your project

Launch Rational Tester for SOA Quality
If you haven't already, launch Rational Tester for SOA Quality (hereafter referred to as Rational Tester). When you launch Rational Tester, you will be prompted to supply a location for your workspace. A workspace is the top level of container for all of your work. You can select any location you'd like for this workspace.

Create a project

Why a performance test project?
You may find it strange that your first step in testing a Web service is to create a performance test project. The reason for doing this is that Rational Tester for SOA Quality is built on the Rational Performance Tester framework. This framework gives Rational Tester many of the fundamental services for Web service testing, such as code-free test design, datapooling, automatic data correlation, and a very powerful reporting structure. Accordingly, here and there along the way you'll see little clues that expose references to the underlying framework.

A project is where Rational stores all of your tests, datapools, schedules and results. Within a workspace, all of your actual work is stored in projects. A workspace can have multiple projects. Follow these steps to create your project (also see Figure 1):

1. Select File > New > Performance Test Project.

2. In the Project name field, type Web Service Tests, and then click Finish to complete the process.

3. Rational Tester will create your project and launch the test recorder to begin testing. At that point, click Cancel. You'll do a little more setup before you begin creating the test.

Figure 1. New Rational Tester project
Import the Web service definition file

A Web Service Definition Language (WSDL) file provides the interface information for the Web service. It defines the functions and specifies the input and output parameters and types. Before you can test a Web service, you need to import this definition into your project by following these steps:

1. The WSDL file for your Web service is in the tutorial .zip file (see Downloads). Go to this section now, and save this file to your hard drive. Expand the .zip file to any location on your hard drive so that you can access the individual files.

2. In Rational Tester, select File > Import.

3. Expand the General folder and select File System.

4. Click Next.

5. In the From directory field, click Browse and navigate to the directory where you saved the ConvertTemperature.WSDL file. In the screen capture in Figure 2, the file is stored in the C:\Documents and Settings\IBM Rational\My Documents folder.

6. In the right-hand display box, select the ConvertTemperature.wsdl file.

7. In the Into folder field, click Browse and navigate to the Web Service Tests project root folder.
8. Click **Finish** to import the WSDL file into your project (Figure 2).

**Figure 2. Import the WSDL file**

---

**Section 4. Create a test**
In this section, you'll create your first Web service test. The Web service that you will use converts temperatures from Fahrenheit to Celsius. You'll call that service two times: First, you'll convert from Celsius to Fahrenheit, and then you'll do the reverse.

Start the recorder

**Animated demo**
To see these steps demonstrated for you, just click:

Show me

1. Start the recorder by selecting **File > New > Test from Recording**. This brings up the **Create New Test from Recording** dialog.

2. Select **Web Service Recording using the Web Services Explorer** (Figure 3), and click **Next**.

**Figure 3. Web Services Explorer in record mode**
3. In the **Test File Name** field, type **CovertTempTest**, then and click **Next**.

4. Click **Add** to add your WSDL file to the test. In a more complicated test, you could provide many WSDL files to test many Web services in a single test.

5. Select the **ConvertTemperature.wsdl** file in your **Web Services Tests** project (this is the WSDL file that you imported earlier in this tutorial). Then click **OK**.

6. Click **Next** to view the Port and Timeout settings. Leave these at their defaults for this test.

7. Click **Next** to view the Privacy Agreement. This agreement specifically warns you that Rational Tester will be monitoring your interactions with the Web service to record the test. Click **I accept** to acknowledge the warning.

8. Click **Finish** to start recording.

Perform the test

**Other recording options**

There are many ways to create tests in Rational Tester. In this tutorial, you'll create your test by using the Web Services Explorer. This method creates a GUI interface based on your WSDL file, which enables you to interact with your Web service. If there is already an HTTP- or Java™-based interface to the Web service, Rational Tester can create a test by using that interface, instead. Finally, for the code savvy, you could manually create your Web service interactions by entering the data required by the Web service directly into Rational Tester.

1. Click the **ConvertTemp** link, as Figure 4 shows.

**Figure 4. ConvertTemp link in Web Services Explorer**
2. In the **Temperature (double)** field, type **20**.

3. In the **FromUnit** field, select **degreeCelsius**.

4. In the **ToUnit** field, select **degreeFahrenheit**.

5. Click **Go** to test the temperature conversion Web service.

6. Your **Status** window should look like the Figure 5.

**Figure 5. Temperature Conversion Status window**

7. Now, convert the 68 degrees Fahrenheit back into Celsius. In the **Temperature (double)** field, enter **68**.

8. In the **FromUnit** field, select **degreeFahrenheit**.

9. In the **ToUnit** field, select **degreeCelsius**.

10. Click **Go** to test the temperature conversion Web service.

11. Click **Stop Recording** under the **Recorder Control** tab to stop recording (see Figure 6).

**Figure 6. Recorder control - Stop Recording**
12. If all has gone well, you should now be looking at your first Web service test (see Figure 7).

Figure 7. Completed temperature conversion test

Section 5. Edit a test

With test creation complete, now you can look at doing some modifications to the test. In the next portion of this tutorial, you’ll modify our test to use a datapool. A datapool is a table of test data, which will enable us to run the test several times with different values.

Animated demo
To see these steps demonstrated, just click:

Show me

Import the predefined datapool
To save typing and time, we’ve created a datapool of values for this exercise. Take the following steps to import the datapool into your project:

1. Select **File > Import**.

2. In the **Import** dialogue, select **General > File System**, and then click **Next**.

3. In the **From** directory field (Figure 8), click **Browse**, and browse to the location where you have stored this file. The **TempValues** datapool is included in the tutorial resources .zip file, where you can also find the WSDL file (see **Downloads**).

4. Select the **TempValues.datapool** file in the right-side selection box.

5. In the **Into folder** field, click **Browse**, and select the **Web Services Tests** project.

**Figure 8. Import the TempValues datapool**
1. Click Finish to import the datapool.

Dynamic data for temperatures

Examine the datapool
You can see the contents of the datapool by double-clicking on the Tempvalues.datapool file in your Web Service Tests project tree. The datapool contains 20 Celsius temperatures, from 5 to 100, and
Modify your test so that Rational Tester pulls values from the TempValues datapool during the test, rather than the value (20) that you entered when you created the test:

1. Double-click the ConvertTempTest tab to maximize the test view on your desktop.

2. The Test Contents panel contains the steps of your test. Select the second line in the box, which reads `ConvertTemp(20, degreeCelsius, degreeFahren)`. This line represents the call to the Web service to convert 20 degrees Celsius to Fahrenheit.

3. The Test Element Details panel (Figure 9) contains all of the details specific to the selected step. Note, specifically, that on the Overview tab, the values are shaded in light green. This indicates that these input values are candidates for datapool substitution.

4. If you were to run your test now, it would submit the values that you see on this tab. The goal, however, is to vary the temperature provided (in this case, 20 degrees) by using a datapool. To do this, first click the cell that contains the word Temperature, and then click the cell that contains the value 20. Click the Test Element Value button, which appears on the right (see Figure 10).

Figure 9. Test Element Details panel

![Test Element Details panel](image)

Figure 10. Test Element Value button
5. In the Edit dialog, right-click 20 and select Substitute From > Datapool Variable.

6. On the Select datapool column dialog click Add Datapool.

7. On the Import Datapool dialog ensure the TempValues.datapool file is highlighted and click Select to select file.

8. In the Select datapool column dialog, select the Celsius column.

9. Click Use Column to have the test use the values from this column of the datapool instead of the 20 you entered manually when you created the test.

10. Click Ok to close the Edit dialog. Note that on the Overview tab, the Temperature value is now in dark green shading. This indicates that, at test playback time, the test will pull values from a datapool rather than using the recorded literal value of 20 (see Figure 11).

Figure 11. Temperature datapool

Use a verification point to validate the Web service response

An important step in test automation involves the insertion of verification points. Verification points are points in the test where you have Rational Tester validate the response received from the Web service. Take the following steps to ensure that the response you get from the Web service is correct:

1. If it is not already maximized, double-click the ConvertTempTest tab to maximize the test view on your desktop.
2. The **Test Contents** panel contains the steps of your test. Select the third line in the box, which reads `ConvertTemp("68")`. This line represents the response from the server, 68 degrees Fahrenheit, that you received when you asked the Web service to convert the 20 degrees from Celsius to Fahrenheit.

3. Right-click this third line, and select **Add > Equal Verification Point**. You'll see the verification point branch on your test (Figure 12).

**Figure 12. Verification point**

4. On the **Overview** tab in the **Test Element Details** panel, expand the `ConvertTempResponse` branch to expose the `ConvertTempResult` branch. (See Figure 13.)

**Figure 13. Associate a datapool with your verification point**

5. Because you'll be using a datapool to vary your input temperature, your output temperature will also vary. You'll need to tie your verification point to the datapool. Click the `ConvertTempResult` variable name, then click the 68 value.

6. A "..." button will appear. Click this to edit your verification point.

7. In the **Edit** dialog, right-click 68, and select **Substitute From > "Fahrenheit" variable, of TempValues datapool.**
Examine automated data referencing

Recall that, when you created the test, you made two Web service calls. The first converted a value from Celsius to Fahrenheit and the second did the reverse. At this point, you have modified your test to use dynamic test data from the datapool for your first Web Service call. Now, let's examine why you don't have to do the same for the second call.

1. Click the fourth line of the test, which reads `ConvertTemp ("68", "degreeFahren...", "degressCelsius").`

2. In the Test Element Details panel, click the Overview tab.

3. Note that, this time, the Temperature line is shaded in purple (Figure 14). This indicates that Rational Tester has performed automated data referencing, sometimes called data correlation on this variable. This means that, during the test, instead of using the literal value that you see here, it will use a value that is dynamically received from the server.

![Figure 14. Automated data referencing on Fahrenheit temperature](image)

Let's find out which value it will use.

4. Click the Temperature variable name, then the 68.

5. A "..." button will appear. Click this to edit your verification point.

6. In the Edit dialog, right-click 68, and then select Go To > /Envelope/Body/ConverTempResponse/ConverTempRes....

7. Click OK to clear the Edit dialog.

At this point, you are looking at the data referencing source. This is the value that will be submitted to the second Web call during the test. In your test, that focus has
moved to the second line, which is the response to your first Web service call. During the test, Rational Tester will use the value received from the server in response to your first Web service call as the value sent to the server for your second Web service call. Because you are already using a datapool on your first value, this response will always be different. Thus, your second Web service call will also always be different.

This is great news. You don't need to do anything to add your second Web service call to the datapool, because Rational Tester spots the relationship between your two Web service calls and ties the two together for you. This automated data referencing feature will save you tons of time when you create complex sequences of Web service calls. It also helps ensure that your tests remain accurate, even when you start to vary your input data with a datapool.

One final verification point

Your last step is to insert a final verification point on the second Web service call. A logical step here would be to tie this value back to the original datapool. This would ensure that the final conversion is equal to the initial value submitted. But given that you already tied your first verification point to the datapool, you'll do something a little different for this one.

One of the most powerful features of Rational Tester is its ability to use regular expressions to match patterns of data. In the next few steps, you'll validate the value returned by your second Web service call. However, instead of validating the actual temperature returned, you'll do a less stringent validation: You will validate that the value returned is a one- or two-digit number.

1. Right-click the last line of the test, which reads `ConverTemp('20')`.
2. Select Add > Equal Verification Point.
3. In the Test Element Details panel, click the Overview tab.
4. Expand all branches so that you can see the last branch (Figure 15).

Figure 15. Final verification point
5. Click the **ConvertTempResult** branch.

6. Click **20** in the **Value** column. Click the ... (ellipses, or three dots) button that appears in the cell.

7. In the **Edit** dialog, enter `[0-9]{1,2}`. This is a regular expression that, when translated to English, means: "Accept any one- or two-digit combination that uses the numbers 0 through 9." Or, put another way: "Accept any number between 1 and 99."

8. In the **Edit** dialog, click **OK**.

9. Click in the **Regexp** column. This will place an **X** in the column, indicating that the entry in the **Value** column is a regular expression.

**Summary**

At this point, you've done the lion's share of the work. You've recorded a Web service test that makes two Web service calls. You've modified your test to use a datapool, meaning that you can run this test with an infinite amount of input data. You've also created two verification points to validate Web service responses. Your first verification point ensures that the Web service response is an exact match to what you were expecting, while the less stringent second verification point ensures that your response is a number between 1 and 99.

Your next steps will be to include the test in a schedule and then to analyze the results of the test execution.

---

**Section 6. Schedule test execution**

**Examine the datapool**

You can see the contents of the datapool by double-clicking on the **Tempvalues.datapool** file in your **Web Service Tests** project tree. The datapool contains 20 Celsius temperatures, from 5 to 100, and their Fahrenheit equivalents.

Your next step will be to create a schedule for test execution. This will enable you to sequence multiple tests and to control the number of test iterations.

In the next few exercises, you will create and execute a simple, single test schedule.
Run individual tests

In this next section, you'll formally schedule a test run. This will enable you to loop through the test and use multiple values from the datapool. However, if you want to run just a single instance of your test without scheduling, you can.

In the Test Navigator, right-click the ConverTempTest item in the directory tree, and select Run As > Performance Test.

Your test will run, using the first value in the datapool. This is a quick and efficient way to test your Web service.

Create a schedule

A schedule specifies which tests to run and in which sequence. Here, you'll create a schedule that loops your test five times, using the first five values in your datapool. Following these steps to create your schedule:

2. In the Performance Schedule dialog, click your Web Service Tests project to select it as the parent folder.
3. In the Name field of the Performance Schedule dialog, enter the name: Web Service Test Schedule.
4. Click Finish to create the schedule. You will see the beginnings of your schedule.
5. On your schedule, right-click the User Group 1 (100%) branch, and select Add > Loop.
6. In the Schedule Element Details section, under Number of iterations, enter 5.
7. On your schedule, right-click the Loop (5 iterations) branch, and select Add > Test.
8. In the Select Performance Tests dialog, select the ConverTempTest that you created in the first section of this tutorial.
9. On your schedule, click the top branch of your schedule, labeled Web Service Test Schedule.
10. In the Schedule Element Details section, as the Number of users, enter...
11. Select File > Save to save your work. Your schedule should now look like Figure 16.

Figure 16. Completed schedule

Summary

That was simple, wasn’t it? In this case, it certainly was. Scheduling execution can be as simple or as complex as you’d like. For this purpose, we just wanted you to create a simple schedule that would quickly run your test. A more complex example could have included more tests, random selection, distributed execution, and complex system resource monitoring options. To learn more about these advanced scheduling options, you can run through the Rational Performance Tester tutorial called Hello World: Learn how to discover and analyze performance issues using Rational Performance Tester (also linked to Resources).

In the next section you’ll execute your test schedule and examine the results.

Section 7. Execute and analyze

At this point, you’ve created your test and you’ve scheduled it for execution. For the most part, your work is done. From here on in, Rational Tester will launch and execute the test and present you with the test results.

Animated demo
To see these steps demonstrated for you, just click:

Show me
Execute the test

1. Executing the test is very simple. Just following these steps to launch the test:

2. In the Test Navigator view, right-click the Web Service Test Schedule and select Run As > Performance Schedule.

3. The Launch Schedule dialog will appear as Rational Tester prepares to run your test, then you will start to see live results from your test accumulate in the Web Service Test Schedule - Web Services Performance Report view.

4. Your test is complete when you see the status turn to green in the Overall tab of the view, as Figure 17 shows.

Figure 17. Completed test schedule execution

Analyze Test Results - Functional

You can analyze test results from a functional test or a performance test
perspective. From a functional test perspective, the objective is to ensure that the
Web service returned the proper results for a given set of inputs. The test log is you
main means of ensuring that you received the correct results. The test log gives you
detailed information about every message sent and received between Rational
Tester and the Web service. To analyze your results:

1. In the **Performance Test Runs** view, right-click the **Web Service Test
Schedule [Time Stamp Information]** and select **Display Test Log**.

2. The test log includes two tabs: **Overview** and **Events**. The log opens to
the **Overview** page, where you will see the pass or fail verdict and the
timing information for the test. Your log should look something like Figure
18.

**Figure 18. Test log overview**

![Test Log Overview](image)

3. Click the **Events** tab to see more detailed information.

4. Expand the **Events** tree so that it looks like Figure 19.

**Figure 19. Expanded test log events**

![Expanded Test Log Events](image)
5. The test log structure is based on the schedule that you created earlier. In this case, expanded to this point, you are looking at the results of the first loop iteration executed by your single user in the group labeled User Group 1.

6. Highlight the first line of your test, which reads: `ConvertTemp (20, degreeCelsius, degree Fahrenheit...)_1`. If all has gone well, in the Properties section of your log, you should see this message: `WebServices called successfully`.

7. At this point, you may wonder why your test looks like it was attempting to convert 20 degrees from Celsius to Fahrenheit. Didn't you hook this up to a datapool with a first value of 5 degrees? To answer this question, you'll need some additional information. From the Main menu, select Window > Show View > Other > Test > WS Protocol Data.

8. Click again on the test line that reads `ConvertTemp (20, degreeCelsius, degree Fahrenheit...)_1`, and examine the data in the new WS Protocol Data view. It should look like Figure 20.

**Figure 20. WS Protocol Data view**
9. Here, you see the exact data that was submitted to the Web service. We can see that you did, indeed, attempt to convert from 5 degrees Celsius to Fahrenheit, as noted in the line that reads `<q0:Temperature>5</q0:Temperature>`.

10. Focus on the next line in the log, which reads `ConvertTemp("68")_1`, and examine the new data in the WS Protocol Data view. Note that the result of the conversion is 41, as noted in the line that reads `<ConvertTempResult>41</ConvertTempResult>`.

11. Look at the next line in the log, which reads Equal Verification Point_1. Note that the text in the properties panel now reads VP Passed. Also note that the WS Protocol Data view shows the actual data returned form the Web service on the left and the expected data contained in the verification point on the right. The two match, letter for letter.

Figure 21. Verification point
12. Finally, examine the next three lines. The first is the call to the Web service to convert 41 degrees Fahrenheit back to Celsius. The second is the response and the third is the regular expression-based verification point that ensures that the response is a number between 1 and 9.

13. Feel free to examine the rest of the test log. What you will find is the same Web service being called over and over, but using the different data from the datapool. The values used should be 5, 10, 15, 20, and 25 degrees Celsius.

Analyze Test Results - Performance

In the previous section, you examined the test results from a functional perspective, primarily ensuring that the system returned the right output for a given set of inputs. Your concern was functional accuracy.

In this next section, the focus switches to performance timing. You'll focus in this section on ensuring that there are no bottlenecks in the system that would prevent the system from responding in a timely manner.

Root cause analysis

In this next section, you'll look to find performance bottlenecks. The first question anyone asks when they find a bottleneck is "Why is this slowdown occurring?" Root cause analysis is a feature of the tool that allows you to dig deeper into the system to determine if the cause is hardware-related, such as a memory shortage or network or CPU utilization problem, or software-related, such as inefficient code. This tutorial does not cover that feature, but if you would like to learn more about it, you can run through the Rational Performance Tester tutorial titled Hello World: Learn how to discover and analyze performance issues using Rational Performance Tester (also linked to Resources).
Modify the schedule for multiple virtual testers

1. If they are still open, close your test log and your results from the previous test execution.

2. Now open your Web Service Test Schedule.

3. Click the top branch of your schedule, labeled Web Service Test Schedule.

4. In the Schedule Element Details section, for the Number of users, type 5.

5. Select File > Save to save your work.

6. In the Test Navigator view, right-click the Web Service Test Schedule and select Run As > Performance Schedule.

7. The Launch Schedule dialog will appear as Rational Tester prepares to run your test, then you will start to see live results from your test accumulate in the Web Service Test Schedule - Web Services Performance Report view.

8. Your test is complete when you see the Complete status turn to green in the Overall tab of the view.

Analyze the performance reports

Note:
From this point onward, your results will not match the results in this tutorial, because performance response times will vary for every run. You are, after all, working against a public Web service that anyone can access at any time. Therefore, we include screenshots of our results and explain them. You should be able to apply a similar analysis to your numbers. If you wish to follow along with these exact reports, the tutorial resources in Resources includes a completed tutorial project that contains the results from this performance run.

1. Examine the Overall tab that Figure 22 shows.

Figure 22. Performance Results - Overall tab
The overall report gives a high-level summary of the accuracy of the test. In our case, we see that 96% of the Web service calls were successful, and that 94% of our verification points have passed. Are these good enough numbers? That depends on your organization's tolerance for faults. Are you OK with 4% of your customers not getting a response to your Web service? In our case, for a noncritical, simple service such as our temperature conversion service, the answer is probably yes. A consumer of the Web service who did not get a result would probably just try the Web service again. However, if this were a more time-critical operation or a data-critical operation, a 96% service level may not be acceptable.

2. Examine the performance results summary that Figure 23 shows.

Figure 23. Performance Results - Summary tab
This tab shows summary information. From this, you know that our test took 1 (one) minute and 27 seconds to complete. There were a total of 5 active users who made 50 Web service call attempts (5 users X 2 calls per user X 5 loop iterations = 50), of which 48 (or 96%, as you saw earlier) were successful. Two Web services timed out while waiting for a response. Our average response time was 2.424 seconds, with our fastest response being .156 seconds. Our standard deviation was a relatively large 3.702 seconds, meaning that our typical response was 2.424 seconds, give or take 3.7 seconds -- although that's a lot of give or take.

3. Examine the **Response Time Results** tab (Figure 24).

**Figure 24. Performance Results - Response Time Results tab**
On that tab, our data is a little more granular than on the previous tab. Here, you see response time per Web service call. For both calls, you see the fastest, slowest, and average response time, as well as the standard deviation. Data is presented graphically and in a table format. Again, you see that our results vary widely, with some very quick responses of shorter than a second and some very slow responses of longer than 9 seconds.

4. Examine the **Response vs. Time Summary** tab that Figure 25 shows.

**Figure 25. Performance Results - Response vs. Time Summary**
This tab shows the average response time for all calls recorded at various points in time throughout the test. Interestingly, here you see that, for most of our test, response times were quite fast, but response times at the beginning and end are quite slow. This explains some of the large standard deviations that you see.

5. Examine the **Response vs. Time Details** tab (Figure 26).

**Figure 26. Performance Results - Response vs. Time Details**
This tab breaks the averages on the previous tab down by Web service call. The squares represent the first call made by every user; the circles represent the second. You can see that the first Web service call was slow at the beginning of the test, and the second was slow at the end. However, because both of our calls are to the same Web service, this breakdown isn't of any great relevance.

6. Examine the **User Action Throughput** tab (Figure 27).

**Figure 27. Performance Results - User Action Throughput**
The top graph of these two graphs shows when users were active. Our numbers here aren't very interesting, given that this was only a five-user run. However, this data can be very interesting for runs with a large number of users, because it will tell you the point at which you had all of your users up and running. The second graph is the call rate. It shows how many Web service calls were made for a given reporting interval. The default interval is 5 seconds, so our graph shows that, at 22.5 seconds into the test, Rational Tester was submitting 5 Web service calls every 5 seconds. Then, just before the 45-second mark, that number jumped up to 10 Web service calls every 5 seconds. This data gives you an idea of the load that we were placing on the server.

Finally, notice that the last tab, the Resources tab, is empty, because we did not collect resource data for this run. Had we been able to do so, we would have been able to measure things such as CPU, network, and disk utilization during the test to determine whether our performance problems had any associated resource constraints. This data is part of the root cause analysis feature of Rational Performance Tester (see sidebar titled "Root cause analysis").
Results

The results are in. We have found that, from a functional perspective, the Web service is working. More specifically, we found that it returned the right output for a given set of inputs.

From a performance perspective, we found that the Web service is working fairly well. Under a relatively small load, we had a few slow responses and a few timeouts, but 96% of our calls went through. Given the nature of our Web service, this is probably good enough, although more testing should be done. After all, our 'load' was only five users who sent a total of only 50 calls in a minute and a half. It would be good to ramp up to a significant number of users and calls to truly validate system scalability.

As you've now seen, Rational Tester for SOA Quality and Rational Performance Tester Extension for SOA Quality are two tools that you can use to validate Web service functionality and performance. In this tutorial, you created -- without writing a single line of code -- a test that validated the functionality of a public, temperature-conversion Web service. Your test included multiple data sets and performed multiple types of response data validation. You then ran simultaneous instances of that test to validate system performance under load. Not bad for an hour's work!

To learn more about either Rational Tester for SOA Quality and Rational Performance Tester Extension for SOA Quality, please visit our Web site or IBM® developerWorks® from the links in the section here titled Resources.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files to use for this tutorial</td>
<td>Tutorial_Resources.zip</td>
<td>4649KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- Watch the Rational Tester for SOA Quality video demo
- Work through the Hello World tutorial: Learn how to discover and analyze performance issues using Rational Performance Tester. Get an overview of Rational Performance Tester in this basic tutorial, which includes both demos and practical, hands-on exercises. You’ll need to download the free trial version of Rational Performance Tester (the link to it is later in this Resources list).
- Check out other parts of the Hello World tutorial series on IBM developerWorks®.
- Visit the Rational Tester for SOA Quality area on IBM.com
- Visit the Rational Tester for SOA Quality area on developerWorks
- Visit the Rational Performance Tester Extension for SOA Quality area on IBM.com
- Visit the Rational Performance Tester Extension for SOA Quality area on developerWorks, where you will find technical documentation, how-to articles, product information, and more.
- Visit the Rational Quality Management home page on IBM.com.
- Visit the IBM Service-Oriented Architecture (SOA) Solutions area on IBM.com.
- In the Rational Performance Tester area of IBM developerWorks, where you will find technical documentation, how-to articles, product information, and more.
- In the SOA and Web services architecture area on developerWorks, get the resources you need to advance your skills.
- Enroll in RT523: Essentials of IBM Rational Performance Tester V7.0 This introductory course focuses on getting started with Rational Performance Tester and practical application of the tool to resolve common performance testing challenges. Testers can build, enhance, and run scripts in a full-function Java Integrated Development Environment (IDE) that integrates with other IBM Rational Software Delivery Platform products. This live instructor-led course takes place online, with hands-on labs and real-time interactions.
- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.
- Subscribe to the developerWorks Rational zone newsletter. Keep up with developerWorks Rational content. Every other week, you’ll receive updates on the latest technical resources and best practices for the Rational Software...
Delivery Platform.

- Subscribe to the Rational Edge e-zine for articles on the concepts behind effective software development.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download a trial version of IBM Rational Performance Tester V7. Choose your language and operating system preferences.

Discuss

- Participate in the Performance Testing forum on IBM developerWorks.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Brian Bryson

Brian Bryson joined Rational Software in 1995 with a strong background in Quality Assurance. Currently, he is a member of the product team that plans and delivers the IBM Rational automated toolset. A former instructor at Carleton University in Ottawa, Canada, Mr. Bryson has taught the IBM Rational approach to quality management to undergraduate students, as well as to employees of organizations worldwide.