

Improved application development: Part 5, Testing and verifying with Rational tools

Skill Level: Intermediate

[Martin C. Brown](#)
IT consultant

Jul 2005

Testing is a vital part of any development process, and to perform adequate testing you need not only to identify faults but also to trace and track these faults, fixes, and the components they affect during each iteration of the development process. In this tutorial, you'll learn about the integration between the IBM Rational software testing products and other tools used in the development process, such as IBM Rational RequisitePro, IBM Rational Application Developer for WebSphere Software, and IBM Rational ClearQuest.

Section 1. About this tutorial

This tutorial is the last in a five-part series. We strongly recommend that you take all the tutorials in this series sequentially, because each tutorial builds on something that was done in the previous tutorial.

The penultimate stage in any development is to test the application. Testing has many facets; you need to test everything from compatibility and performance to whether the application meets the original requirements. Testing can also occur at many levels: testing that occurs during development (often while code is still being written) to ensure the code is working as expected; and structured testing that works within a specific framework and is often used to test against a range of platforms and environments using the same suite of tests.

Using the IBM Rational® tools, you can perform a range of different tests during the development process. Some of these can be handled within the Rational Software Development Platform; some require you to use other software in the Rational suites

to achieve your aims. This tutorial looks at the three main toolsets for testing the Web auction application developed over the course of the tutorial series.

The tutorial includes the following:

- **Test types and tools**
- **IBM Rational Functional Tester:** You'll apply the functional testing mechanism built into the Rational Software Development Platform to the Auction application.
- **Using IBM Rational PurifyPlus™:** Next, you'll use PurifyPlus to provide memory and performance analysis for your program.
- **IBM Rational TestManager:** You'll also learn how to use TestManager, an automated environment for testing across multiple platforms.
- **End of the cycle**

Throughout, this tutorial examines the roll of testing and the information provided by IBM Rational RequisitePro® and IBM Rational ClearQuest®, which are used to help drive the development process.

Prerequisites

The focus of this tutorial is on two products in the Rational Suite®: Rational Functional Tester and Rational PurifyPlus. It also looks at integrating the Rational Software Development Platform (which incorporates the IBM Rational Software Modeler and IBM Rational Application Developer for WebSphere Software components), using this as a basis for testing and deployment. In addition, this tutorial uses ClearQuest and RequisitePro as sources for related information. Links to download trial copies of most these products are listed next.

To complete the steps in this tutorial, you need:

- [RequisitePro Version 2003.06.13](#) or later.
- Rational ClearQuest Version 2003.06.13 or later (contact your IBM reseller)
- Rational ClearQuest Eclipse Client:
 - If you have Rational ClearQuest Version 2003.06.13.x, you need Version 6.0.0x of the client. Download it here:
<http://www-1.ibm.com/support/docview.wss?uid=swg24007802>
 - If you have Rational ClearQuest Version 2003.06.14.x, you must use

the 6.14.x version of the Rational ClearQuest Eclipse client. Download this version of the client by selecting **Help > Software Updates > Find and Install** within the IBM Software Development Platform. Follow the online instructions to install the client.

- [Rational Application Developer](#) Version 6.0.
- [Rational Software Modeler](#) Version 6.0
- [Rational Functional Tester](#) Version 6.1
- [Rational PurifyPlus](#)

If you haven't already completed this step earlier in the series, you also need to install the Auction application sample from the Rational Software Development Platform Showcase samples. This application includes a prebuilt version of the Auction application you've been building in this tutorial series. Using the prebuilt version allows you to continue the tutorial with the remainder of the code already configured. To install the sample:

1. Choose **Help > Samples Gallery** to open the Samples Gallery window.
2. Expand the Showcase samples folder.
3. Expand the Auction Application folder.
4. Expand the Construction folder.
5. Click **Web Application**.
6. Click **Import the sample**. This option imports the code and source files into your current workspace.

Find the EJB components for the application in your EJB Projects folder as the project AuctionV60EJB.

Section 2. Test types and tools

All application software should be tested. No matter how good your developers or your software development processes are, you're going to end up with faults in your applications. Some of these involve errors in logic; others are related to programming errors. Others, of course, are due to the design and requirements of the original application, which won't always match the expectations of the project's

original stakeholders. Testing can also be used to verify the operation of an application against the original requirements.

Testing should account for at least 25% of the development effort; many people argue that 50% or even 80% is a better figure. Obviously, the methods and practices built into the IBM Rational Unified Process® (RUP) are designed to help reduce the number of errors in your code through the use of requirements management and model-based development, frequent testing, and an iterative approach that helps to incrementally improve the overall quality of your software.

What the RUP doesn't aim to do is eliminate the need for the full-coverage functional, performance, and analysis testing that should be part of every development process. The Rational toolset includes an extensive range of testing tools that you can use both individually and integrated with other Rational tools. These testing tools help with the collation of information about the tests, execution of the tests, and providing communication and integration between the test assets (the tests and their results) and the application components.

Monitoring all this information and tracking the effects of the dependencies between different components and the tests executed on those components can be tricky. This is where the Rational software excels.

This tutorial focuses on the testing process: how to test components and how to collate and report the information using the auction example as a guide. The role of requirements in the testing process is also examined, as well as how to integrate testing information between the testing applications and Rational ClearQuest, the defect-management tool.

Integrating testing with development

It's inevitable that you'll test components during the development process. This is a natural way of ensuring that your code works and compiles to the basic level you expect. Using Rational Application Developer (RAD) and the Rational Functional Tester applications, test the functionality of your application to ensure it meets the needs of the project's stakeholders. Use this method also to ensure that your requirements specification matches the final application.

Many teams divide application development from application testing as if the two processes are completely separate. Testing is seen as a process that occurs at the end of development, designed to identify bugs and ensure that the application meets requirements. Often this approach means that a large percentage of time is devoted to development and a small percentage is devoted to the testing part of the process.

Within the RUP, testing is an integral part of development. Unlike the situation just outlined, testing occurs throughout the development process and as part of each

development iteration. Problems identified during each iteration can then be fixed and resolved as part of the next iteration.

This incremental style combines the development and testing phases and enables you to continually improve and monitor the quality of your software. In the short term, this approach produces better quality releases. In the longer term, it spreads the testing phase over the entire period of development, effectively increasing the amount of time spent testing without pooling the entire sequence at the end of development.

This process also has the effect of improving your time to market, because it helps reduce the time between the initial safe version and the final release.

Feeding off requirements and change requests

One of the most important aspects of the integration between the Rational software components is the ability to use the requirements for the Web Auction system you generated in [Part 1](#) of this series to drive the tests that verify whether the application achieves its functional specification. This link between the original requirements generated in RequisitePro and the tests you create and use to verify the application is critical to ensuring that your application is of the highest quality.

Requirements and the tests you create based on their definition fall into both of the main test categories: functional testing and performance testing. *Functional tests* match requirements with the application's abilities. *Performance tests* match the required performance to the achieved performance. Outside of the requirements specification, use other tests to examine your application. These can be conducted on a component-by-component basis and can be used to identify potential bugs like memory leaks or to provide component-specific performance testing.

Because you can view the requirements and change requests from within RAD, it's easy to see what needs to be fixed and updated. You can also use the information provided by RequisitePro and ClearQuest to guide you during the testing phase. These requirements and change requests form the basis of the tests -- from functional testing of the application to ensure it meets requirements to performance testing to ensure the application runs efficiently.

Let's start by looking at how to use Rational Functional Tester to confirm the capabilities of your Web auction application.

Section 3. Using Rational Functional Tester

Rational Functional Tester is a component in the Rational Software Development Platform. Technically, it's another application in the suite, although like the other components such as Rational Application Developer and Rational Software Modeler, it's based on Eclipse™.

As an Eclipse product, it's ideally suited to the testing of Java applications; it also integrates with the Microsoft™ Visual Studio™ .NET environment, making it practical for testing .NET projects as well as traditional Eclipse-based Java™ projects. Eclipse comes with a wide range of testing tools (such as JUnit) for performing specific types of tests. But it's through the plug-in system that the flexibility of the Eclipse platform begins to shine.

Rational Functional Tester expands on this system and incorporates additional testing functionality -- in particular, a very easy to use functional testing component. The functional testing system has many features, but the most useful is the ability to record sequences as if you were using the application itself: the application records your clicks and selections in the background and produces a savable script of the process to use for testing the functionality again and again through the course of the project.

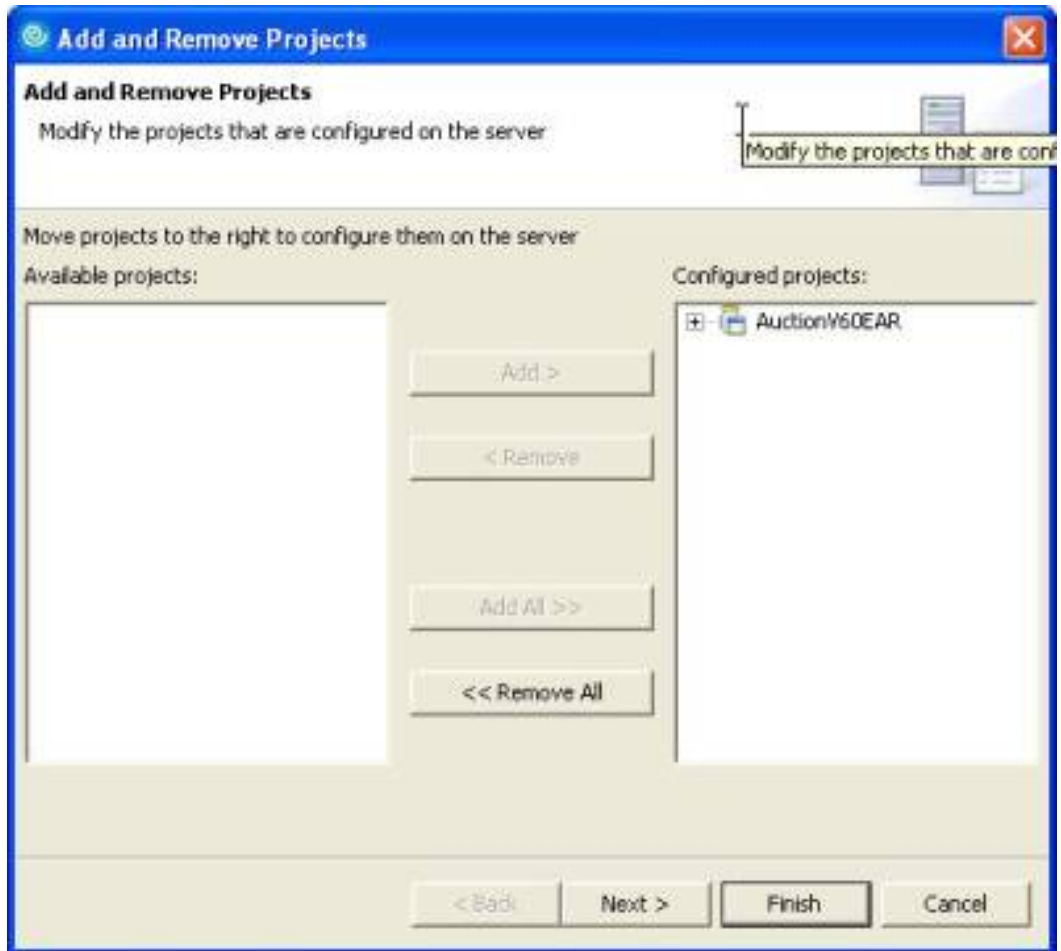
Starting the Web auction project

To use Rational Functional Tester, you must first deploy the Web auction project to a WebSphere server so that the application can be executed as it would be in a real deployment situation.

The deployment is straightforward:

1. Open the server's view by selecting **Window > Show View > Servers**.
2. Right-click the WebSphere Application Server you want to use for testing, and select **Add and remove projects**. You'll see the Add and Remove Projects window (see Figure 1).

Figure 1. Adding a project to a WebSphere server



3. Select the Auction project in the list on the left, and click **Add** to add it to the list of configured projects.
4. Click **Finish**, and wait for the WebSphere server to be populated with the project. It might take some time for the project to be deployed to the server.

To make sure the application is running, in Project Explorer view, expand the Dynamic Web Projects folder, then the AuctionV60Web folder, and finally the WebContent folder. Now select the index.jsp item, right-click, and select **Run on Server**.

The Auction project is now available on the server. If you're using the WebSphere host, the address should be `http://localhost:9080/AuctionV60Web/faces/index.jsp`.

Alternatively, you can copy and paste the Web address given when you elected to run the project .

Recording a test script

Now that your Web application is running, you need to record a test script that will be used to test a particular piece of functionality. Before you get to the specifics, let's look at how the system works.

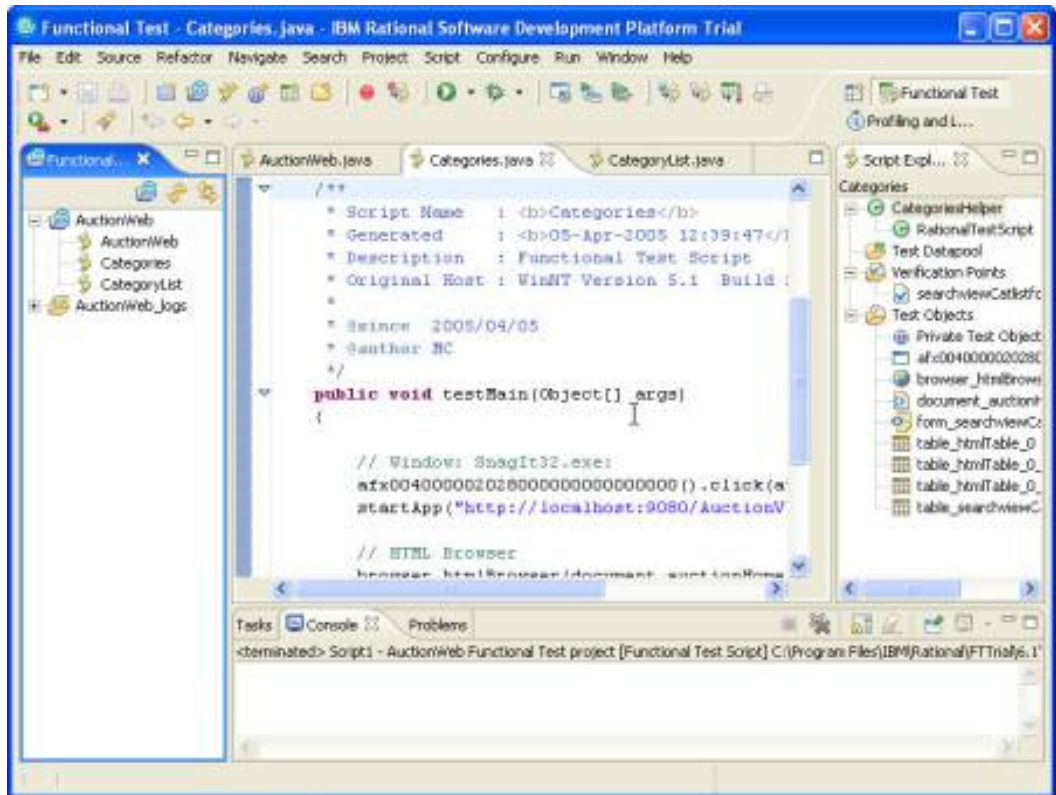
The scripting process is designed to automate the sequence of events you'd experience as an application user; all you're doing is saving the steps so that they can be executed programmatically by the functional testing environment.

When you record a test script, Rational monitors the steps you take, including (in the case of a Web application) what pages you view, the links you click, which options you select, and other steps you follow, in order to build a script of events. During the process, you can set expected values and results so that the script both tests the application and confirms its results.

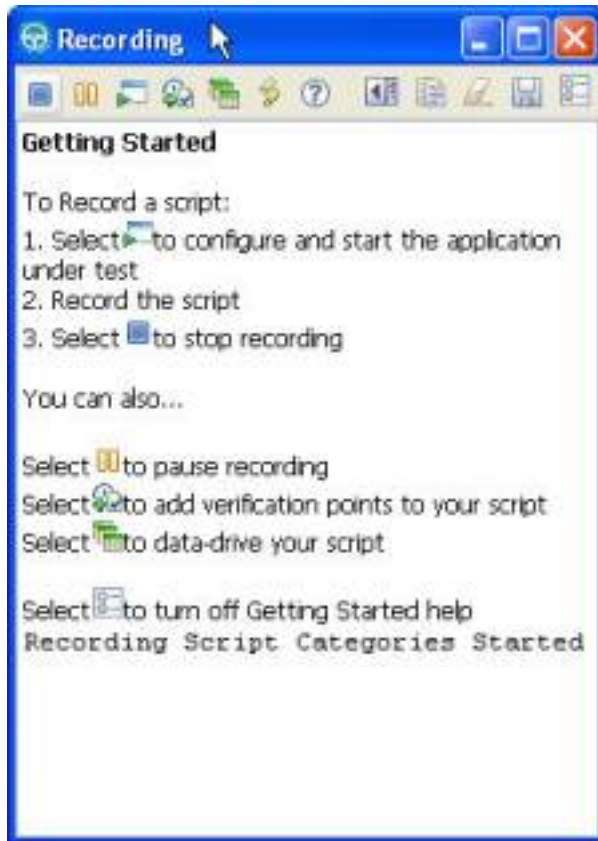
To test your Auction application, you're going to create a test to ensure the component that displays a list of items in the auction system works correctly when you click a category from the home page. Follow these steps:

1. Open the Rational Functional Tester for the application type (see Figure 2); in this case, you're testing a Java script. The basic layout is similar to other Eclipse-based products.

Figure 2. Rational Functional Tester



2. You might need to create a new Functional Test Project. To do this, select **File > New > Functional Test Project**. You'll be prompted with a wizard asking for the location and name of the project.
 3. Select **Script > Add Script Using Recorder**.
 4. Provide a location to save the script and a name for the script. You can select other options at this stage, but the default options are fine; click **Finish**. The Recording window opens (see Figure 3).
- Figure 3. Recording window**



5. To start the Auction application, click **Start Application** in the toolbar of the Recording panel (third button from the left). A window opens in which you select the application you want to start. If the application isn't in the list, click **Edit Application List** and use the form provided to add a new application to the list, using the URL of the Auction application. Once the application is configured, select the application from the list and click **OK**. The home page of the Auction application opens (see Figure 4).

Figure 4. Auction in action

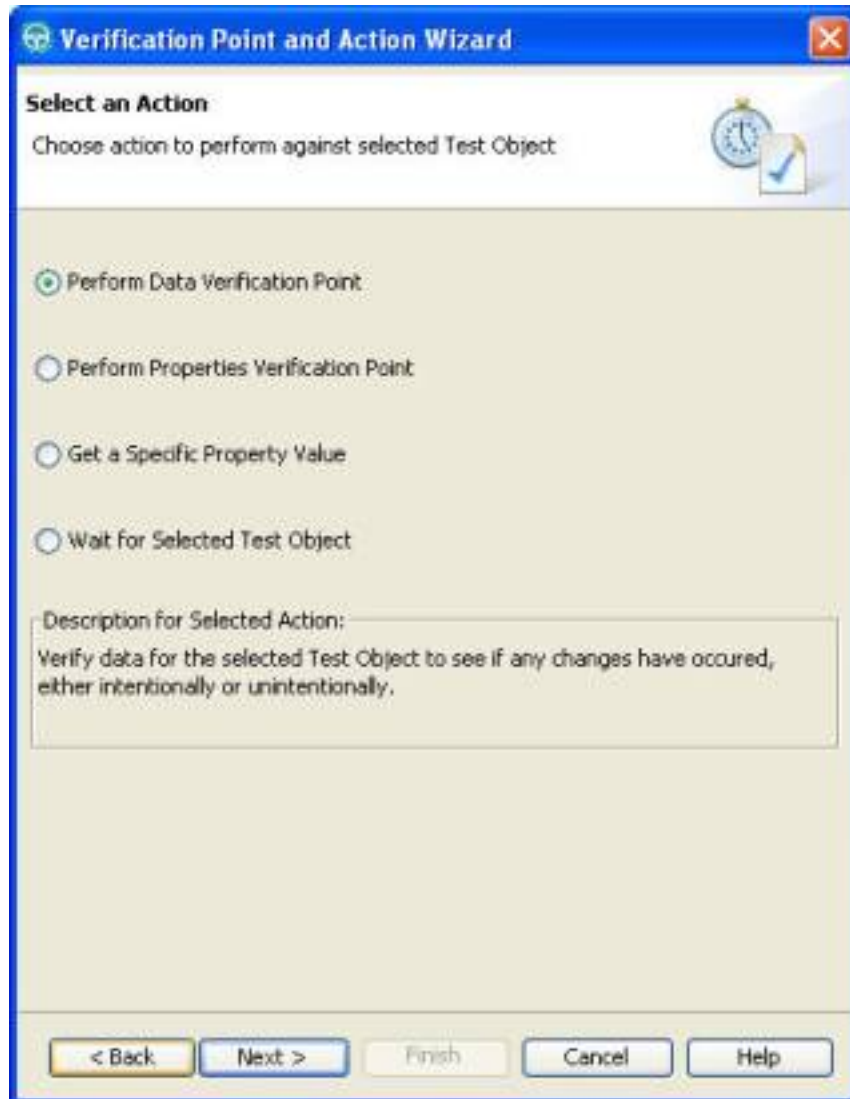


6. To add verification points to ensure that the list of categories on the homepage is correct, click **Insert verification point** in the Recording panel (fourth button from the left). The Verification Point and Action Wizard window opens (see Figure 5).

Figure 5. Verification Point and Action Wizard window

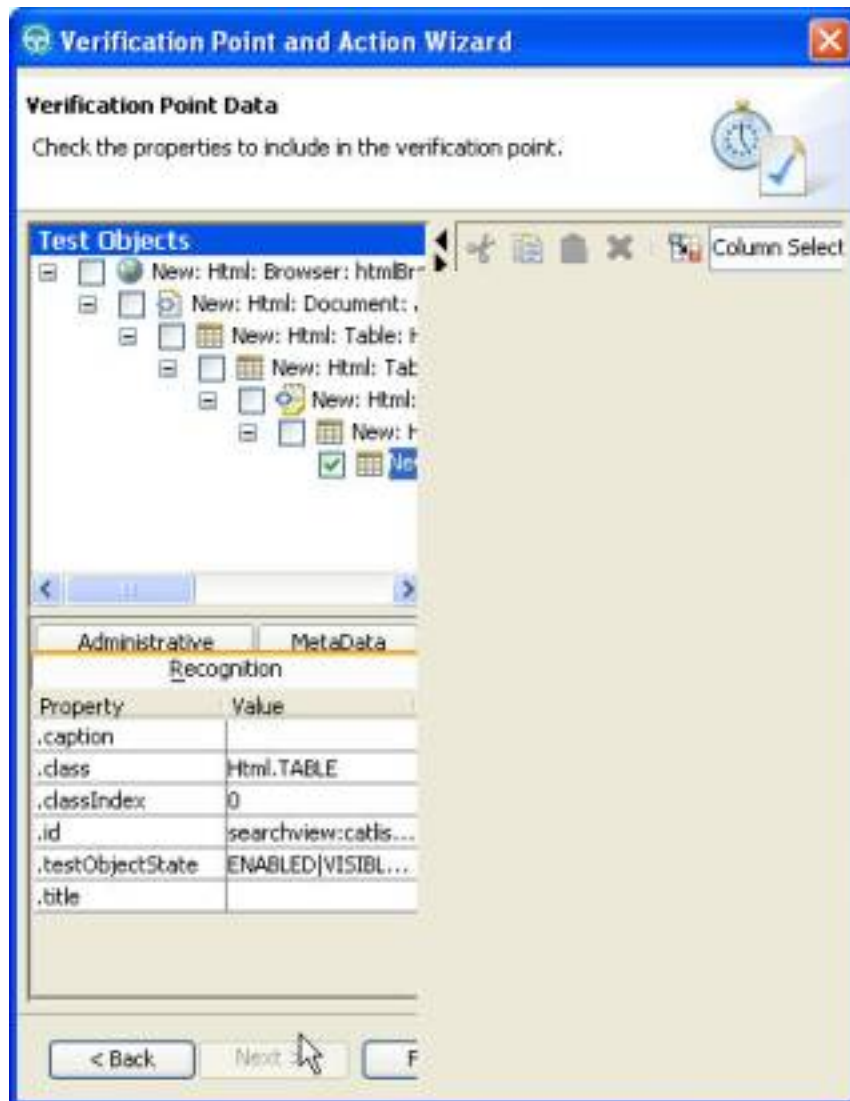


7. Click and drag the **hand button** from the wizard window over the Categories table in the browser window. Notice that a red border is added to components of the system as you move over the browser window. Make sure you select the entire table, not just the heading or contents. Release the mouse button and select an action to perform (see Figure 6).
Figure 6. Selecting a verification action



8. You can select a variety of actions, including checking the properties of a component or a specific property value. For this example, stick with a data verification point. Click **Next** to select the data point. You've already selected the data point by using the hand tool (although you can be more specific or define a wait period for a dynamic component to be populated). Click **Next**.
9. Verify the data you selected (see Figure 7). The wizard automatically fills in the current contents of the table you selected. You can add more points to this data as required. For now, assume that the current listing is correct, and click **Finish**.

Figure 7. Setting the data to verify



10. You're finished with this part of the verification. Click **Stop** in the Recording panel (first button from the left) to finish the recording; the final script appears in the Rational Functional Tester window.

You can use the same basic sequence to record other functional items, including selecting items and making bids. To do so, use the recording system to record your progress through the site and the information you enter, and then use the verification point to confirm that the expected response equals the actual response.

The script you created becomes your permanent testing component for checking the functionality of the system at any later stage.

Running the test script again

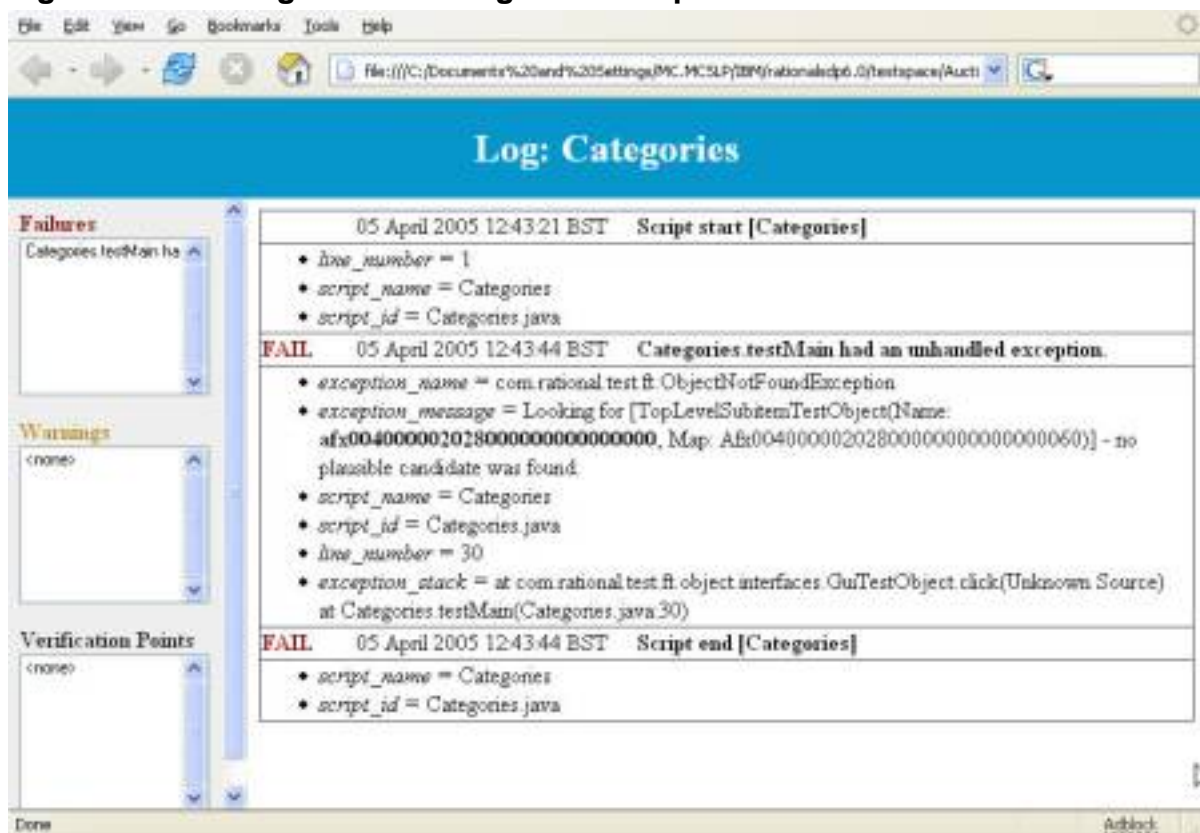
The point of recording test scripts is that once they're recorded, you can use them to repeat tests at a later stage. This is useful to test known issues reported in the ClearQuest defects register and to confirm that the functionality of the system meets what was defined in the RequisitePro requirements generated and refined through this series.

Over the long term during project development, test scripts can also ensure that the core functionality hasn't changed between development cycles. Obviously, some components are expected to change; but once a component or system has reached maturity, it's highly likely that the functionality will remain the same. Use the scripts you create to confirm that the system still operates the way you expect.

To re-execute a test script, select the script from the Functional Test Projects explorer and choose **Script > Run**. The script runs, following the exact sequence you recorded and performs any necessary tests.

As part of the process, Rational Functional Tester generates a log that reports progress and any problems with the script (see Figure 8). Use this information to report defects into ClearQuest.

Figure 8. Error Log when running a test script



You should have multiple test scripts to handle and examine the functionality of a range of different parts of the system. You can have as many of them as you like to

test as many areas as you like. Furthermore, because the technology isn't limited to static results, you can test different sequences and their results. This makes testing much more than simply providing input values and responses for different components: use it to examine the actual user experience.

Section 4. Using Rational PurifyPlus

Rational PurifyPlus provides a suite of tools that can analyze and monitor application development during startup. The suite consists of three tools:

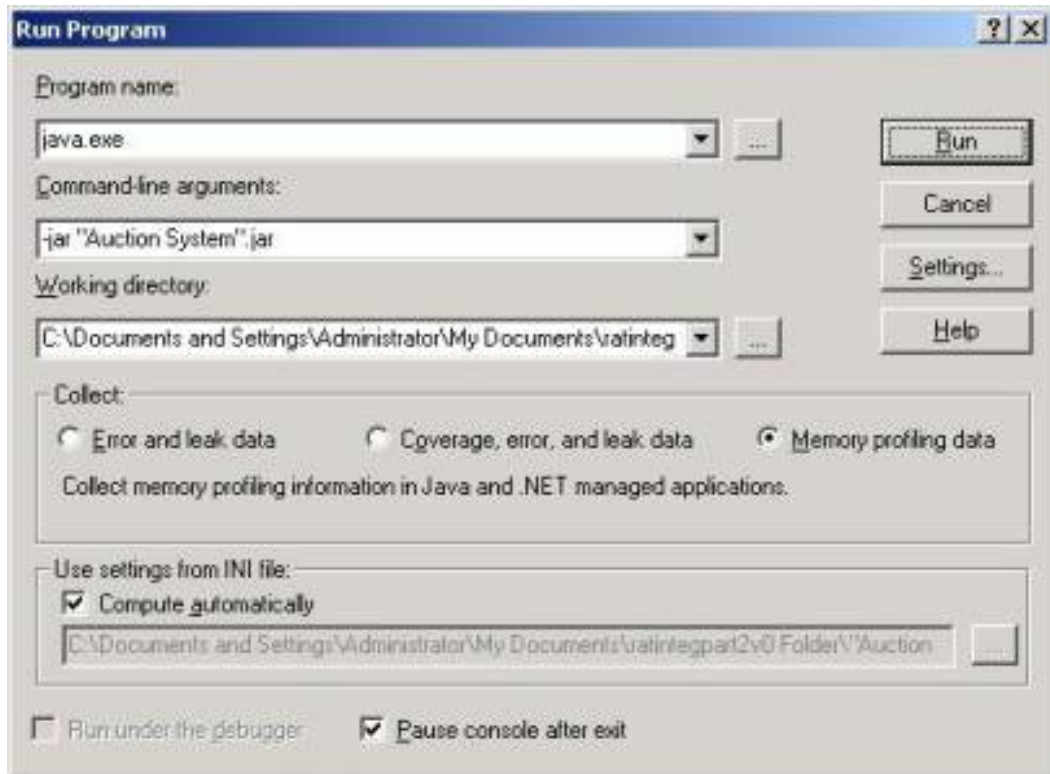
- **Rational Purify** finds run-time errors and memory leaks in your application.
- **Rational Quantify** provides performance analysis on your application, building a profile for your application's startup so that you can identify performance constrictions.
- **Rational PureCoverage** ensures that your entire code base is thoroughly tested.

These tools work individually and together to ensure that your application is tested for basic errors, performance, and code errors.

Using Rational Purify to monitor memory usage relies on taking an initial reading of your application's typical memory usage to provide a baseline. You can then execute the code where you suspect a memory leak. Comparing the memory usage between helps you to determine which methods might be causing memory problems. Armed with information about the methods, examine the objects and class definitions that are the root of the problems. These details are beyond the scope of this tutorial, but this section examines the basic process.

To use Purify on your application, follow these steps:

1. Run your Java program with Rational Purify. Rational Purify provides a simple interface for starting your applications (see Figure 9).
Figure 9. Rational Purify interface for starting an application



2. On each startup, you get a code launch profile. You can see the progress of the application and obtain detailed information about the startup process through the Data Browser view, which includes four tabs:
- **Memory** shows the memory usage (see Figure 10).
 - **Call Graph** demonstrates the execution sequence (see Figure 11).
 - **Function List View** shows the complete list of functions in the application, the number of calls, and object memory (see Figure 12).
 - **Object List View** shows a list of the objects in the application (see Figure 13).

Take a snapshot of the memory during normal use for a portion you don't suspect of causing a problem.

Figure 10. Memory tab of the Data Browser view

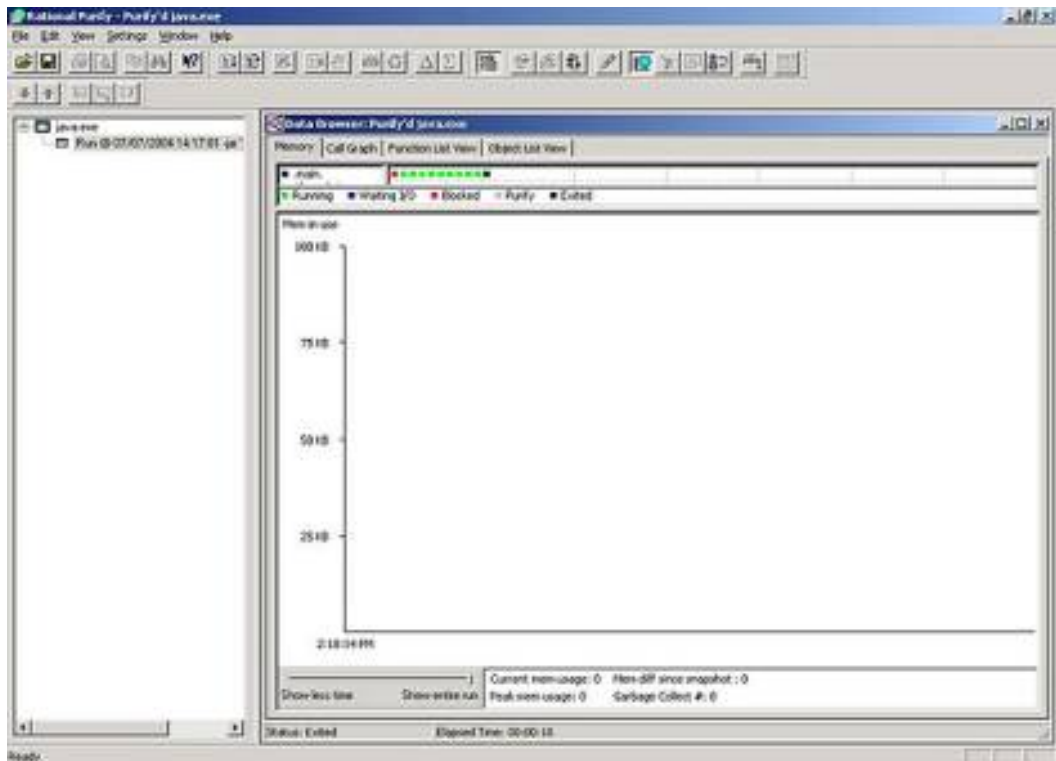


Figure 11. Call Graph tab of the Data Browser view

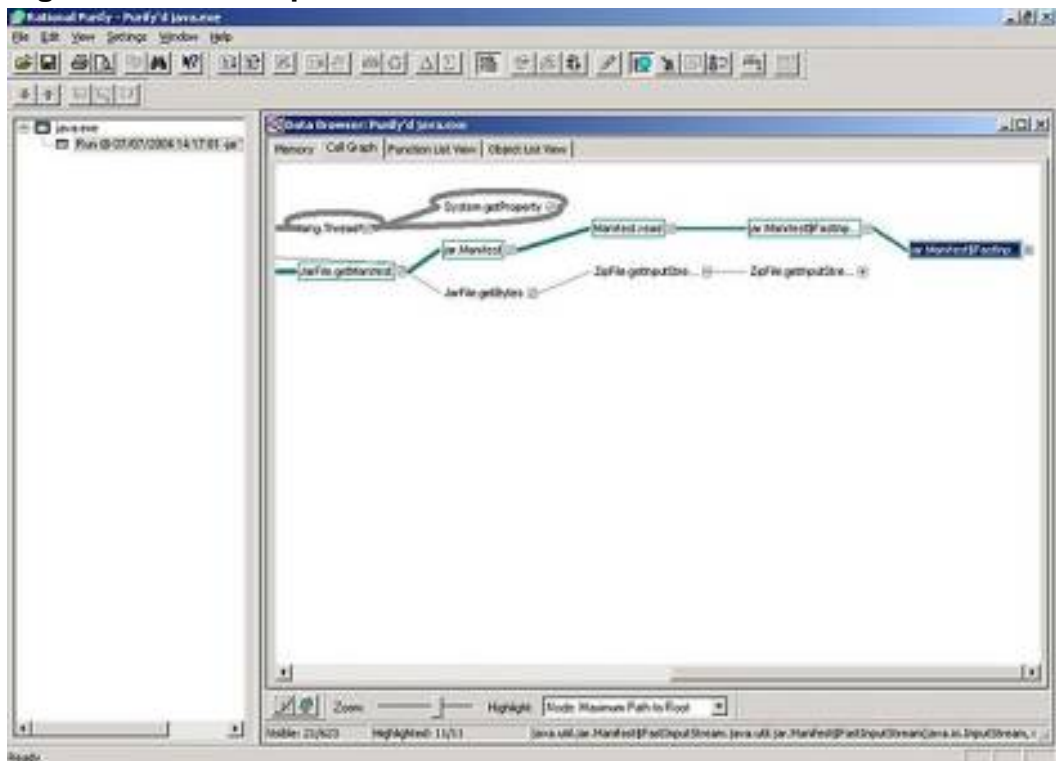


Figure 12. Function List View tab of the Data Browser view

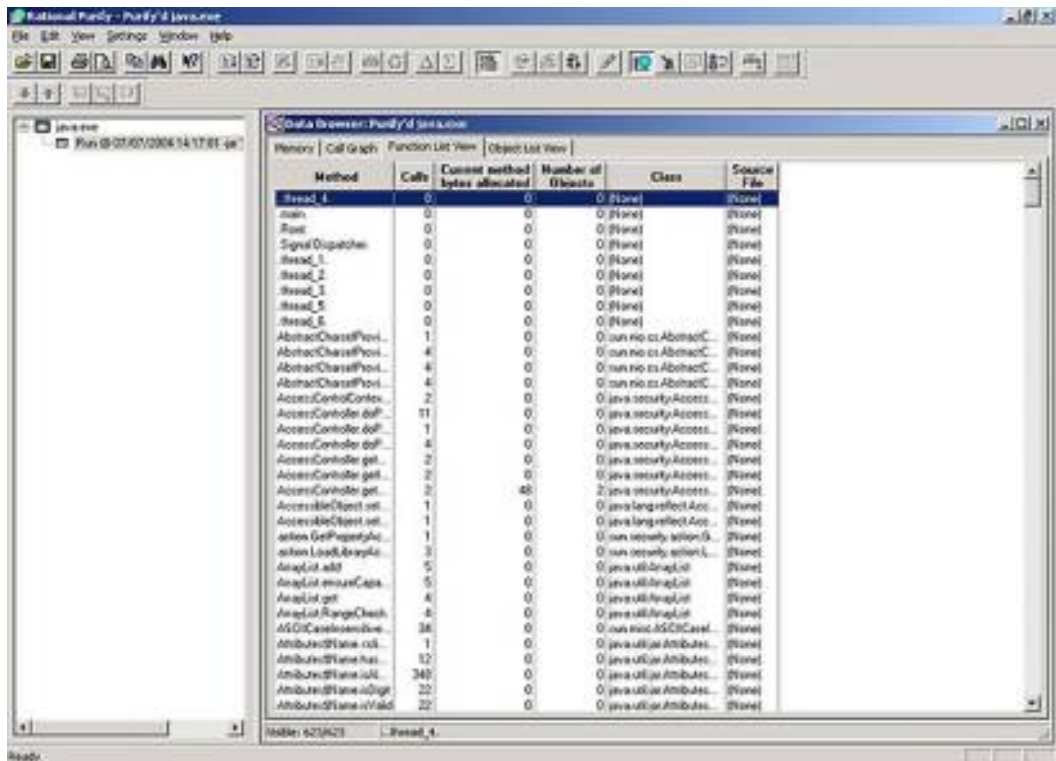
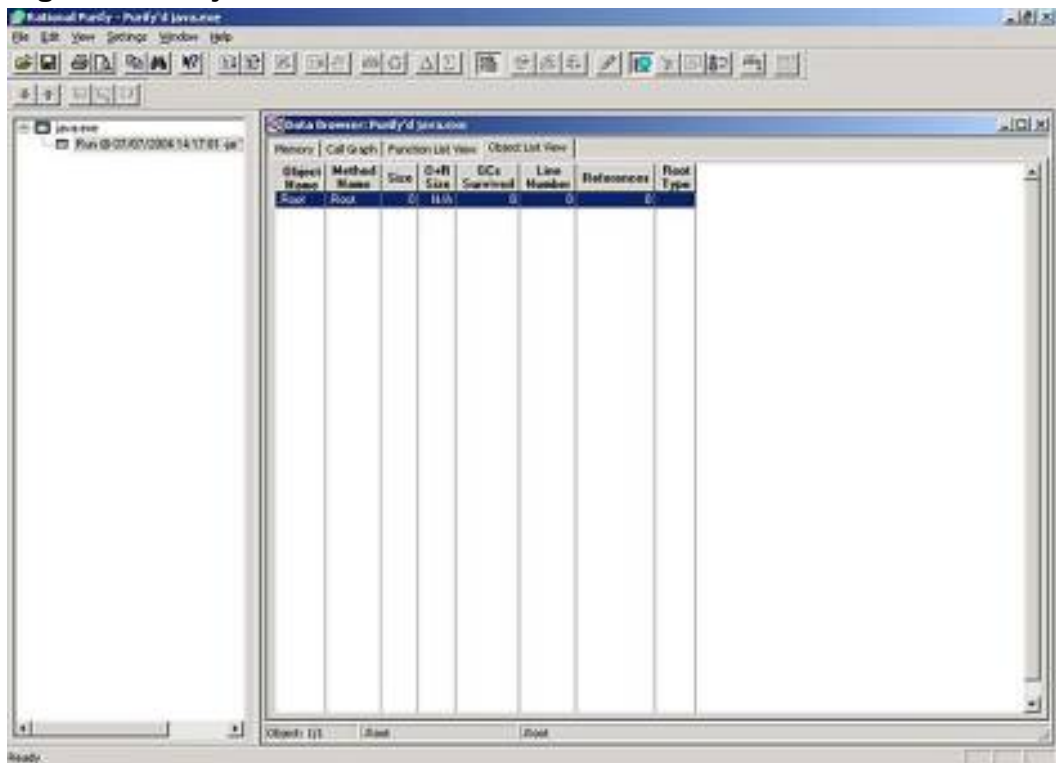


Figure 13. Object List View tab of the Data Browser view



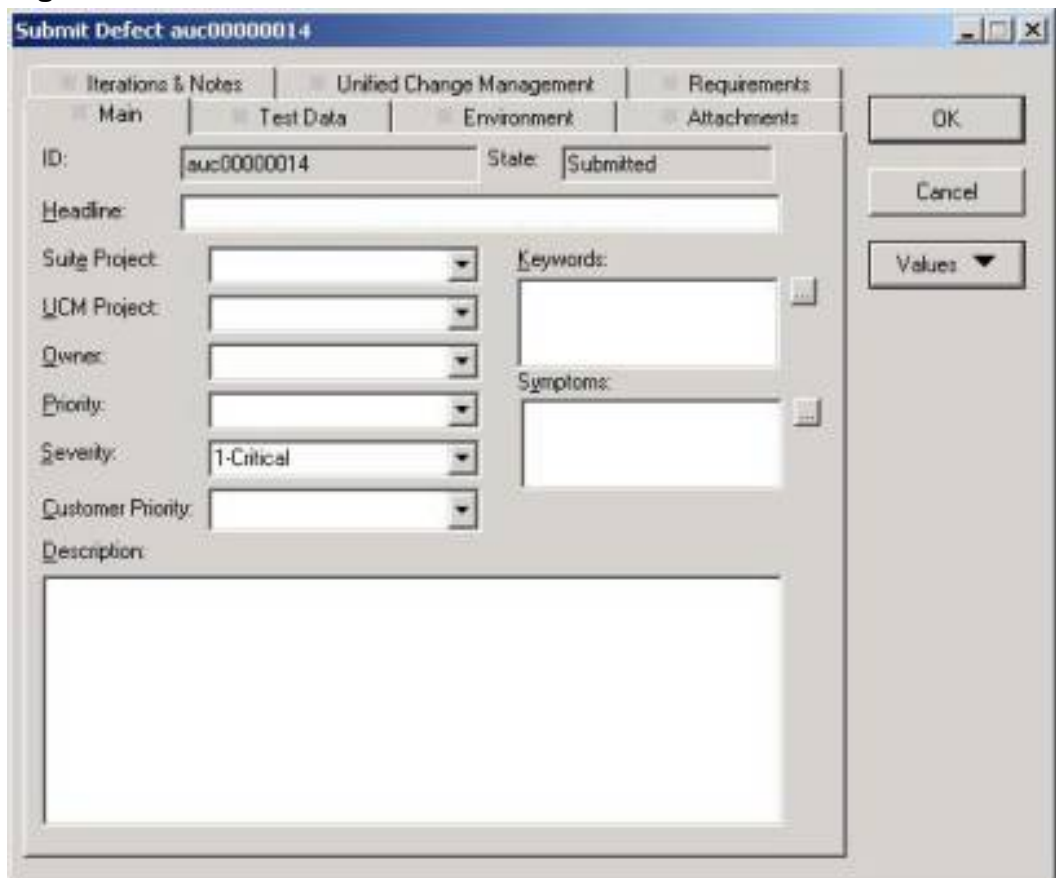
3. Run the code that might be causing memory leaks.

4. Compare the two snapshots -- standard and suspected memory leak -- to determine which parts of the application might be causing a problem.
5. Pinpoint the objects that are causing the problem, and investigate them more fully to determine where the leaks are occurring.

When you find a fault, use the defect submission system. To submit a defect into Rational ClearQuest from any Rational PurifyPlus application:

1. In an Error view, select from the error report all the text you want to submit as the description for the defect.
2. Right-click the text, and select **Submit ClearQuest Defect**. A Submit Defect window opens (see Figure 14) prefilled with the selected text.

Figure 14. Submit Defect window



3. Add any additional information, such as priority and severity, to the form. If you've generated Purify data and reports and saved that information in a file, attach it to the defect entry.

4. Click **OK** to submit the defect.

Performance testing with Rational Quantify

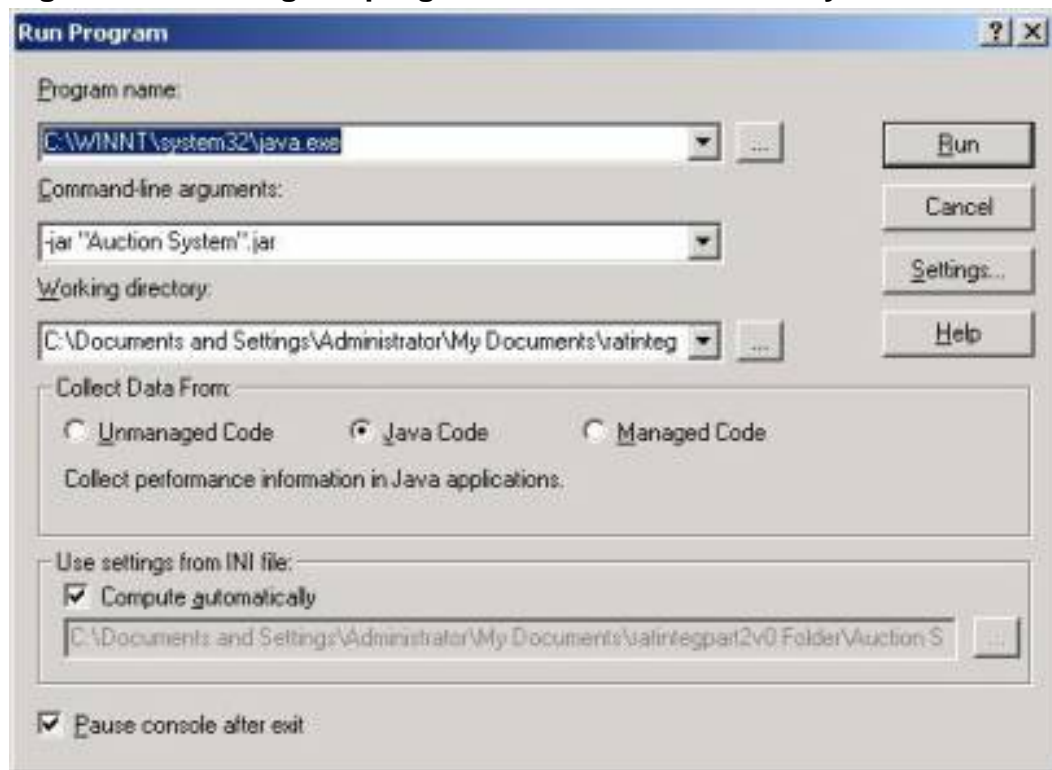
Rational Quantify is all about finding performance constrictions in your application. A typical application has numerous places where a constriction can cause a problem. In the Auction application, performance is vital because users expect a faster response from a Web site than they do from a typical desktop application. Finding these constrictions can be difficult.

Finding constrictions is best achieved through profiling that monitors the time spent executing different components of an application. By studying the timing and execution of these components, you can identify which parts of the application are taking the most time.

To use Rational Quantify for profiling:

1. Run the program with Rational Quantify to collect performance data (see Figure 15).

Figure 15. Running the program with Rational Quantify



2. Use the performance data to identify which components are taking the most time.
3. Modify the code to reduce the constrictions.
4. Rerun the program, and compare your new results with the original tests.

Results can be reported directly into Rational ClearQuest from Rational Quantify in the same way as from Rational Purify.

Coverage analysis with Rational PureCoverage

The purpose of Rational PureCoverage is to ensure that you've fully tested all the components of your application before it's released to your clients and customers. The problem is to analyze which components make up the bulk of your application and thereby identify which components will benefit from the bulk of the testing process. Rational PureCoverage also identifies which components and lines of code in the application haven't been run and therefore haven't been subjected to the full range of tests.

You should use Rational PureCoverage throughout the development process to monitor the tested components. Using the data generated by Rational PureCoverage as you go through each iteration, you can determine which components are new in the system and which deserve more attention as you test and monitor the application.

When you execute Rational PureCoverage on your application, you can identify the following:

- Each function defined in the application but not tested
- The number of lines of code that have been missed (not executed)

With this detail, you can identify the application components that should be tested more stringently. If you identify a coverage problem, submit the defect directly into Rational ClearQuest from Rational PureCoverage:

1. In the Module view of the File View tab of the Data Browser window, select the file or function with the coverage defect that you want to report.
2. Right-click the file or function, and choose **Submit ClearQuest Defect**. As before, the Rational ClearQuest Submit Defect window opens.
3. Add any other information that you want to record for this defect.

4. Click **OK** to submit the defect.

Rational PurifyPlus is only one part of the testing process, though. You've already seen the functional testing platform provided by the Rational Software Development Platform; other tools, such as Rational TestManager, provide structured testing across multiple platforms and environments.

Section 5. Rational TestManager

The Auction application is Web based, but this doesn't mean you can rely on it to work across a range of platforms and environments. Unfortunately, the Internet and Web browsers aren't the utopia of platform independence you may expect. Whether you're using Internet Explorer, Netscape®, Mozilla™, or Firefox™, you have a range of different versions. On the server side are various deployment platforms from Sun™, Apache™, IBM, and many others. Again, there are a range of different versions and, worse, a variety of combinations.

Testing all these combinations by hand is time consuming. Furthermore, testing all the components and possible input and output values for all the different components by hand could take days. Process automation is helpful.

Rational TestManager is a complete solution for testing applications; it makes up elements of Rational Functional Tester for Java and the Web, Rational Robot, and Rational Performance Tester. Rational TestManager is designed to work in a team environment as a central testing solution; you can build suites of tests that can then be executed on an application at any stage during development and during any iteration of the development process.

Each time you run a test, the results are recorded and stored. Use this information to develop an idea of the faults in your system and, over the long term, to record and monitor statistics related to bugs in the application you're developing. For example, by monitoring and tracking this information, you might find that one part of the application is generating the most errors and therefore requires additional attention during the debugging process.

TestManager also provides a distributed testing facility, allowing you to spread the tests for different components of an application across a number of machines. Use this facility to execute a small number of tests and parameters across an application in a shorter time or to provide a much wider range of parameters and therefore more extensive tests, without the test process taking days or weeks. Distributed testing also enables you to test across a range of platforms using the same test inputs and

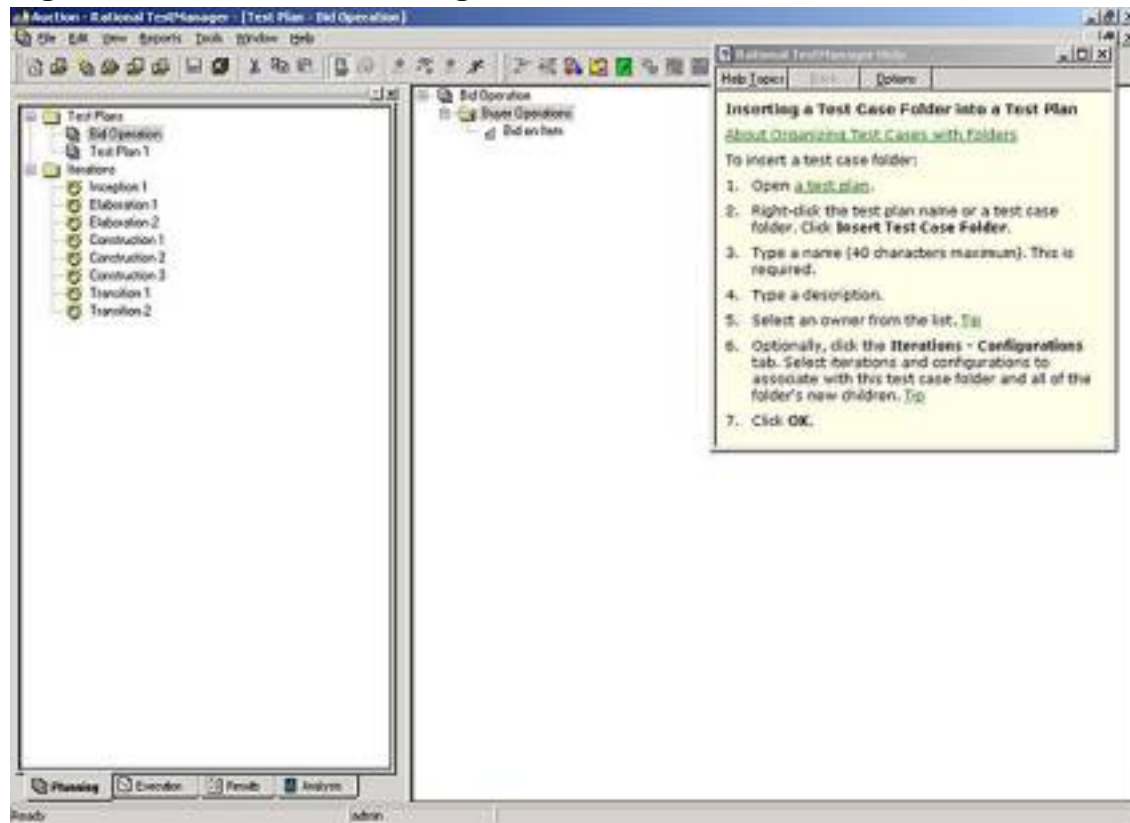
components and monitor the results from a central location.

Once all the information from the components has been collated, Rational TestManager provides a convenient way to report and summarize the information. This is useful to all the groups involved with the project. Project analysts, architects, developers, quality assurance personnel, and project managers can all use the information provided by TestManager to gauge the status of the project.

Inside Rational TestManager

Figure 16 shows a sample window from Rational TestManager.

Figure 16. Rational TestManager



The main window is split into two sections. The panel on the left provides an interface to the various parts of the Test project. The tabs at the bottom switch between the sections used to drive the test process:

- **Planning** provides an interface to the test plan structure and the iterations (including any parameters) used to test those components.
- **Execution** is the interface for controlling the execution of the tests.

- **Results** provides the results for each test execution.
- **Analysis** is the interface into analyzing and collating raw results into meaningful reports on the status of the tests.

The panel on the right displays details about individual components -- for example, the properties of an individual test or the results from a test. This panel also displays other windows used during the testing and management process, such as a list of test inputs and reports and results.

TestManager testing process

Tests in Rational TestManager are centered around and controlled by *test plans*. Creating a test plan enables you to define the test's parameters in terms of test objectives. A test plan isn't a static entity: Use a test plan throughout the life of a project to define and build new tests and new situations as the project progresses.

A test plan consists of a list of *test cases*. You can organize individual test cases within a test plan in a hierarchical fashion to make it easy to organize and identify the different tests in the system.

Building a test plan involves a number of steps. To create a series of tests in TestManager, follow this sequence:

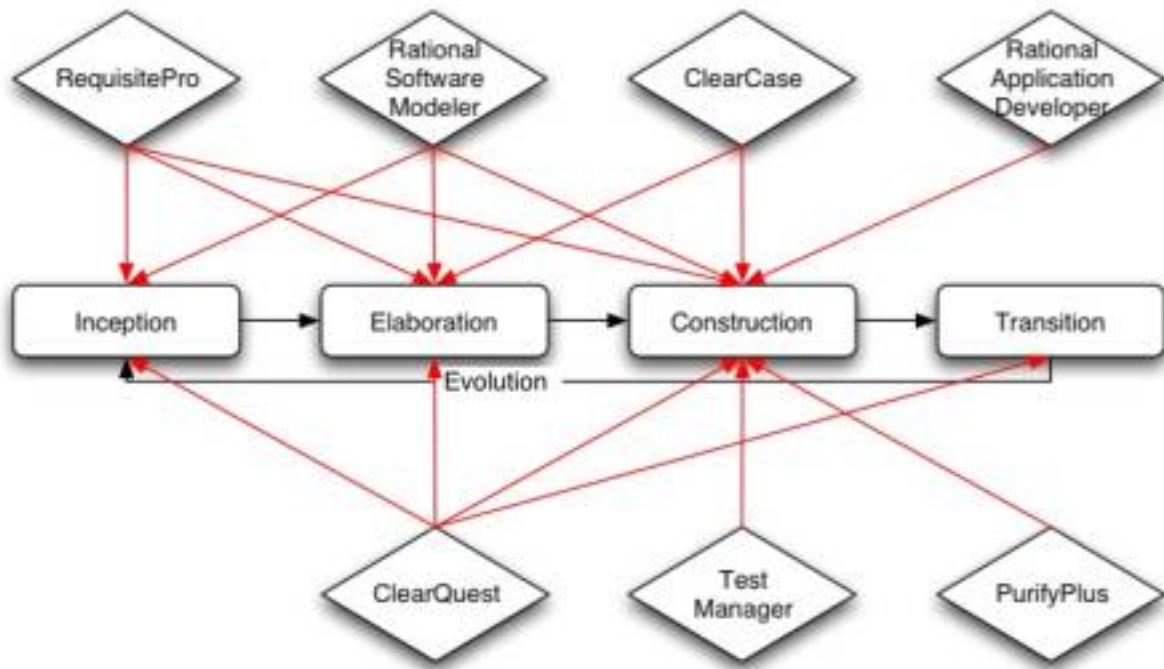
1. Define *test inputs* that define the entities tested by the test cases in a test plan.
2. Create a test plan to hold the information about input, test cases, and other components.
3. Create test cases that define the individual tests to be performed. You can organize these into test case folders for easier identification.
4. Define the configurations. *Test configurations* are the OS environments you want to use when testing the system. In a Web environment, such as your Auction application, configurations are used to define the server computers that will be accessed and the client computer configurations that will access the server.
5. Define the iterations that define when the tests will be executed.

You now have a structure in place to perform tests across multiple platforms and environments.

Section 6. End of the cycle

Throughout this tutorial series, you've learned about the application development process (see Figure 17) and how you can use the various Rational tools individually and together to produce an application of the highest quality. The tools in the Rational suite have helped through every step of the process. It's worth reviewing the tools, their sequence, and their roles during the development process.

Figure 17. The Rational development process



Having examined the tool set and the development process throughout this series, you can begin to see the sequence and integration that let you share information between components and more easily develop top-quality applications.

Preparing the next iteration

You've followed the process from the original application requirements; to developing the model and writing the code; to monitoring changes, defects management, and testing. Now that you've achieved this first iteration of application development, repeat the process.

The next iteration improves on this release. The exact sequence depends on the

application, but typically it involves some or all of the following steps:

1. Combine the defects and change requests from Rational ClearQuest with the original requirements to generate the next revision of the requirements specification in Rational RequisitePro.
2. Use the new requirements to adjust the models in Rational Software Modeler.
3. Use the models to generate new code that is merged with the other code in the system to generate the application components.
4. Update the code to fix any defects identified during testing in the previous iteration.
5. Update and expand the tests from the first iteration, if necessary, and re-execute the tests on the new build of the application.

Go through the same process again with each new iteration, expanding the application, improving on the previous build, and fixing the faults and performance problems identified in the previous revision.

Requirements-driven model

It should have been clear from Part 1 of this series that the requirements for the Auction application, as defined by the project's various stakeholders, are the main driving force behind the application's development. These requirements define everything about the development process from a specification for the application to the performance requirements. All of this is managed through Rational RequisitePro.

The requirements also form a vital part of the ongoing development model. The requirements become the linchpin of the process: you can link everything, from the original client requirements and specification to the models and code components to the requirements. Once you have this link, it's easy to connect other components such as test results to the components and ultimately back to the requirements. Through this same system, you can also connect defect requests and changes and enhancements back to the original requirements.

Armed with this information, you can refine and update the requirements to help refine the requirements of the application and provide a guide and description of the application you're trying to build. Ultimately, this leads to better application development. By having a centralized location for controlling the application and recording defects, enhancements, and other components, you can concentrate on the application components that need to be developed, updated, and fixed.

Section 7. Summary

This tutorial covered the role of software testing and the Rational software that provides testing facilities. Rational Functional Tester provides an easy interface for building scripts to test the functionality of your application. It works with a wide range of application types, from stand-alone applications to the Web application you've been using here.

The Rational PurifyPlus suite provides a complete solution for determining internal faults such as memory leaks, performance issues, and other programming problems. For functional testing across a range of platforms and environments, Rational TestManager provides a comprehensive interface to a number of Rational tools, including Functional Tester. Through the Rational TestManager system, define tests and distribute them through a variety of platforms and environments to get an all-around test on as many environments as possible. Using the test cases, you can also develop and define tests that cover your application's entire range of functionality.

You can integrate the test applications with other components in the Rational suite. For example, automatically filter results into the Rational ClearQuest system so you can monitor and report on your tests' progress as they're conducted and recorded.

This five-part series has examined the crucial role Rational software plays in aiding application development. Rational software is a management platform for monitoring the entire development effort, from the inception of the requirements through testing components and ensuring that the developed application matches the client's demands.

Resources

Learn

- [Collating requirements for an application, Part 1](#) is the first part of this tutorial series.
- [developerWorks Rational zone](#) provides many more resources for you.
- The [Rational Unified Process](#) is a structure development model that affects everything from the documentation to the sequence of developer.
- [IBM Software Development Platform](#) product page. Find more resources and overview information.
- The IBM developerWorks article [Reduce complexity with model-driven development, Part 1: Use the IBM Software Development Platform to develop end-to-end solutions](#) shows how to use the tools in the IBM Software Development Platform to create an end-to-end development cycle.
- The [Rational TUP home page](#) provides information about the IBM Rational Team Unifying Platform (TUP).
- The [Eclipse project](#) is an open source IDE that forms the basis of many IBM products, including Rational Software Modeler and Rational Application Developer.

Get products and technologies

- [Rational RequisitePro](#) Version 2003.06.13
- [Rational Application Developer](#) Version 6.0
- [Rational Software Modeler](#) Version 6.0
- [Rational Functional Tester](#) Version 6.1
- [Rational PurifyPlus](#)

Discuss

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

About the author

Martin C. Brown

Martin C. Brown, a StudioB author, is a former IT director with experience in cross-platform integration. A keen developer, he has produced dynamic sites for blue-chip customers including HP and

Oracle and is the technical director of Foodware.net. Now a freelance writer and consultant, MC (as he is better known) works closely with Microsoft as an SME, is the LAMP Technologies Editor for *LinuxWorld* magazine, is a core member of the AnswerSquad.com team, and has written a number of books on topics as diverse as Microsoft Certification, iMacs, and open source programming. Despite his best attempts, he remains a regular and voracious programmer on many platforms and in numerous environments. MC can be contacted at his [Web site](#).