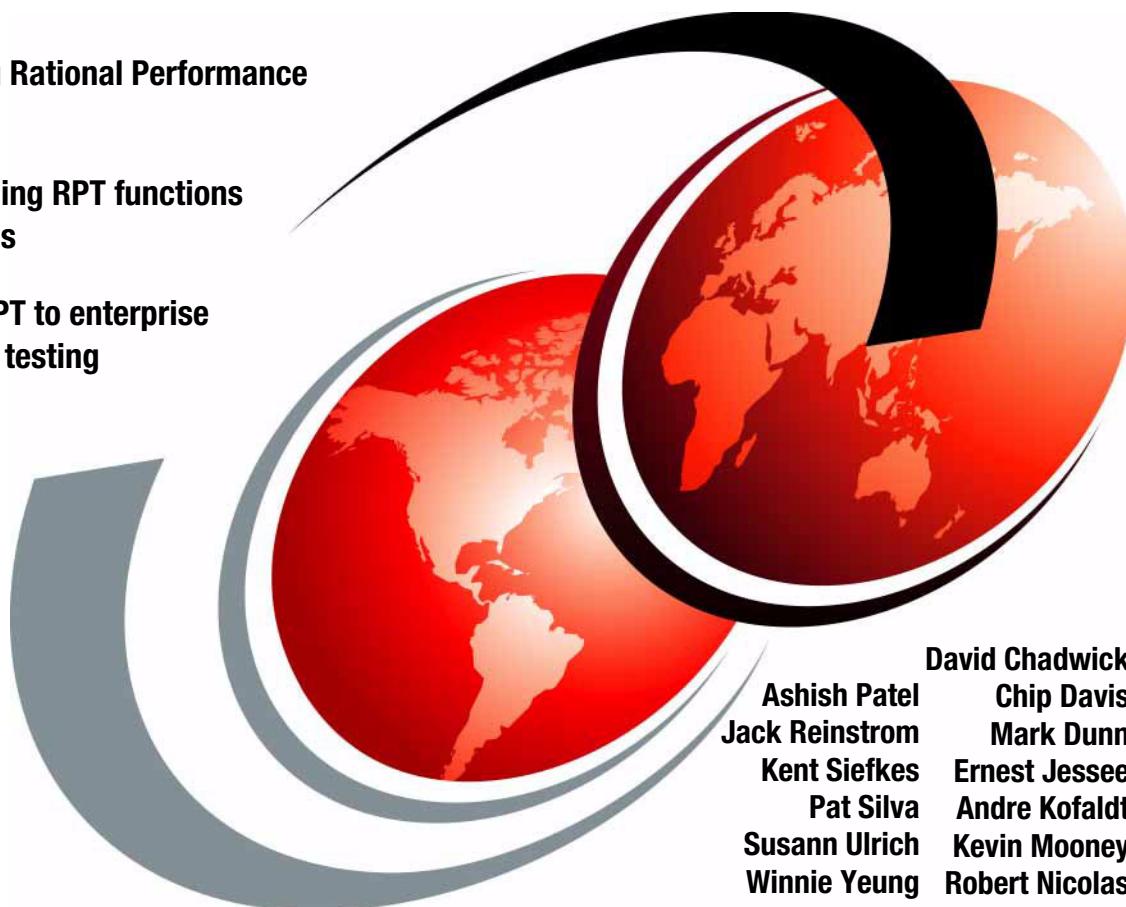


Using Rational Performance Tester Version 7

Introducing Rational Performance
Tester

Understanding RPT functions
and features

Applying RPT to enterprise
application testing



David Chadwick
Ashish Patel
Jack Reinstrom
Kent Siefkes
Pat Silva
Susann Ulrich
Winnie Yeung
Chip Davis
Mark Dunn
Ernest Jessee
Andre Kofaldt
Kevin Mooney
Robert Nicolas

Redbooks



International Technical Support Organization

Using Rational Performance Tester Version 7

March 2008

Note: Before using this information and the product it supports, read the information in "Notices" on page xiii.

First Edition (March 2008)

This edition applies to IBM Rational Performance Tester Version 7.

© Copyright International Business Machines Corporation 2008. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | |
|--|-------|
| Notices | xiii |
| Trademarks | xiv |
| | |
| Preface | xv |
| The team that wrote this book | xvi |
| Become a published author | xviii |
| Comments welcome | xviii |
| | |
| Part 1. Understanding Rational Performance Tester | 1 |
| | |
| Chapter 1. Introduction | 3 |
| 1.1 Overview of IBM Rational Performance Tester | 4 |
| 1.1.1 The performance testing problem space | 4 |
| 1.1.2 Need for a performance testing tool | 5 |
| 1.1.3 Goals of IBM Rational Performance Tester | 6 |
| 1.2 Architecture of IBM Rational Performance Tester | 7 |
| 1.2.1 Eclipse and Java | 8 |
| 1.2.2 TPTP as a tool framework | 8 |
| 1.2.3 Building a complete tool chain | 9 |
| 1.2.4 Recording in a distributed environment | 9 |
| 1.2.5 Scalable distributed playback | 10 |
| 1.2.6 Test run monitoring and data collection | 10 |
| 1.2.7 Post run analysis | 11 |
| | |
| Chapter 2. Performance testing process | 13 |
| 2.1 Establishing a performance testing capability | 14 |
| 2.2 The performance testing process | 15 |
| 2.2.1 Performance test goals | 16 |
| 2.2.2 Workload model of the system's busy hour | 17 |
| 2.2.3 Measurements and metrics | 18 |
| 2.2.4 Test success criteria | 19 |
| 2.2.5 User scenarios in the workload model | 19 |
| 2.2.6 Input data variation in user scenarios | 20 |
| 2.2.7 Debugging user session tests | 20 |
| 2.2.8 Workload model for each experiment | 21 |
| 2.2.9 Test execution results and data collection | 22 |
| 2.2.10 Analysis and tuning | 24 |
| 2.2.11 Performance test final report | 25 |
| 2.3 Performance testing artifacts | 26 |

| | |
|---|-----------|
| 2.3.1 Workload specification | 26 |
| 2.3.2 Performance test plan..... | 27 |
| 2.3.3 Performance test tool assets..... | 27 |
| 2.3.4 Performance test results | 27 |
| 2.3.5 Performance analysis document..... | 27 |
| Chapter 3. Recording and test generation | 29 |
| 3.1 Configuring the browser before you record..... | 30 |
| 3.1.1 Set the application to record | 30 |
| 3.1.2 Verify browser settings | 30 |
| 3.1.3 Set Mozilla environment variables | 32 |
| 3.1.4 Verify that the Agent Controller is running | 33 |
| 3.2 Recording from a secure Web site | 33 |
| 3.2.1 Disabling security warnings when recording..... | 33 |
| 3.3 Non-browser recordings | 34 |
| 3.4 Reconfiguring IE after recording is interrupted | 34 |
| 3.5 Troubleshooting..... | 35 |
| 3.5.1 Agent Controller problems | 35 |
| 3.5.2 Proxy recorder architecture..... | 36 |
| 3.5.3 Interpreting a .REC file (recording file) | 36 |
| 3.6 HTTP recording tips | 37 |
| 3.6.1 Browser caching | 37 |
| 3.6.2 Overlapped requests and responses | 38 |
| 3.7 Test generation | 38 |
| 3.7.1 HTTP test generation | 39 |
| Chapter 4. Test editing and augmentation | 41 |
| 4.1 Overview of test editing | 42 |
| 4.1.1 What you need to know about editing tests | 42 |
| 4.1.2 Editing and augmenting tests versus schedules | 42 |
| 4.1.3 General rules when editing | 43 |
| 4.1.4 Colors and test editor preferences | 44 |
| 4.2 Test editing best practices..... | 46 |
| 4.2.1 Projects and folders | 46 |
| 4.2.2 Test search and replace | 46 |
| 4.2.3 Maintaining variations of a test | 52 |
| 4.3 Editing different types of tests | 53 |
| 4.3.1 Editing HTTP tests | 53 |
| 4.3.2 Editing Siebel tests | 55 |
| 4.3.3 Editing SAP tests | 58 |
| 4.3.4 Editing Citrix tests | 62 |
| 4.3.5 Other types of systems under test | 64 |
| 4.4 Providing tests with test data..... | 64 |

| | |
|--|------------|
| 4.4.1 Providing test data versus data correlation | 64 |
| 4.4.2 Test data considerations | 65 |
| 4.4.3 Datapool overview | 65 |
| 4.4.4 Creating datapools | 66 |
| 4.4.5 Adding datapools to tests | 70 |
| 4.4.6 Substituting test values with datapools | 74 |
| 4.4.7 Summary of datapool and test editing | 77 |
| 4.4.8 Custom test data handling | 79 |
| 4.4.9 Using datapools for digital certificates | 80 |
| 4.5 Manipulating data correlation | 81 |
| 4.5.1 Automatic correlation | 81 |
| 4.5.2 Manual correlation | 83 |
| 4.5.3 Ensuring proper data correlation | 85 |
| 4.6 Adding verification to tests | 89 |
| 4.6.1 Verifying expected behavior in HTTP and Siebel tests | 90 |
| 4.6.2 Verifying expected behavior in SAP tests | 96 |
| 4.6.3 Verifying expected behavior in Citrix tests | 101 |
| 4.6.4 Custom verification | 104 |
| 4.7 Adding test elements | 105 |
| 4.7.1 Inserting and positioning elements | 105 |
| 4.7.2 Delays | 107 |
| 4.7.3 Loops | 108 |
| 4.7.4 Conditions | 111 |
| 4.7.5 Transactions | 114 |
| 4.7.6 Custom code | 116 |
| 4.8 Adjusting connections and timing | 120 |
| 4.8.1 Modifying timing | 120 |
| 4.8.2 Modifying connections | 123 |
| 4.8.3 Changing environments for testing | 128 |
| 4.8.4 Manipulating authentication | 130 |
| 4.9 Improving test reuse and documentation | 135 |
| 4.9.1 Comments and descriptions | 135 |
| 4.9.2 Improving test readability | 137 |
| Chapter 5. Test playback and debugging | 139 |
| 5.1 Test playback | 140 |
| 5.2 Schedule playback | 140 |
| 5.2.1 Create schedule | 140 |
| 5.2.2 Playback schedule | 142 |
| 5.3 Playback status | 143 |
| 5.4 Monitor and control | 144 |
| 5.4.1 Statistics interval | 144 |
| 5.4.2 Change log level | 145 |

| | |
|---|-----|
| 5.4.3 Add virtual users | 146 |
| 5.4.4 Stop test run | 147 |
| 5.5 Debugging | 148 |
| 5.5.1 Problem determination log | 148 |
| 5.5.2 Launch debugging. | 148 |
| 5.5.3 Firewalls and virus detection software | 151 |
| 5.5.4 Test run aborted | 152 |
| 5.5.5 Out of memory error | 152 |
| 5.5.6 Too many files open | 153 |
| 5.5.7 Address in use | 153 |
| 5.6 Command line interface | 153 |
| Chapter 6. Scheduling tests | 155 |
| 6.1 Schedule overview | 156 |
| 6.2 Creating schedules | 156 |
| 6.3 Schedule elements | 157 |
| 6.3.1 User group overview | 157 |
| 6.3.2 Adding elements to schedule | 158 |
| 6.4 Controlling schedule behavior | 165 |
| 6.4.1 Setting think time behavior | 166 |
| 6.4.2 Controlling schedule execution rate | 167 |
| 6.4.3 Controlling user behavior | 167 |
| 6.4.4 Controlling execution duration. | 170 |
| 6.4.5 Controlling log data collection | 171 |
| 6.5 Advanced scheduling topics | 173 |
| 6.5.1 Emulating network traffic from multiple hosts | 173 |
| 6.5.2 Setting problem determination level | 175 |
| 6.5.3 Monitoring resource data | 177 |
| 6.5.4 Monitoring response time breakdown | 178 |
| 6.5.5 Setting statistics displayed during a run | 179 |
| Chapter 7. Results analysis and reporting | 181 |
| 7.1 Introduction to analysis and reporting | 182 |
| 7.2 Statistical basis for reporting performance results | 182 |
| 7.2.1 Random arrivals yield normally distributed sample measurements | 182 |
| 7.2.2 Time correlation | 183 |
| 7.2.3 Steady state | 184 |
| 7.2.4 Confidence interval determination. | 184 |
| 7.2.5 How much data is enough to predict the true response time | 185 |
| 7.2.6 Value of percentiles for response time measurements | 186 |
| 7.2.7 Average values for transaction throughput rates | 186 |
| 7.3 Basic reports and customization | 187 |
| 7.3.1 Correlating multiple results on one report | 192 |

| | |
|--|------------|
| 7.3.2 Analyzing performance within time ranges | 199 |
| 7.3.3 Using percentile reports for response times | 201 |
| 7.3.4 Measuring transaction rates | 203 |
| 7.4 Using reports to find time consuming application steps | 205 |
| 7.4.1 Data collection configuration | 206 |
| 7.5 Some common scenarios for further analysis | 208 |
| Chapter 8. Resource monitoring | 211 |
| 8.1 Overview of resource monitoring | 212 |
| 8.2 Configuring a performance schedule | 213 |
| 8.3 Monitoring using Windows Performance Monitor | 216 |
| 8.3.1 Performance counters | 216 |
| 8.3.2 Data collection | 217 |
| 8.3.3 System configuration | 218 |
| 8.3.4 Configuring a performance schedule | 220 |
| 8.4 Monitoring using rstatd for Linux/UNIX systems | 222 |
| 8.4.1 Performance counters | 223 |
| 8.4.2 Data collection | 224 |
| 8.4.3 System configuration | 225 |
| 8.4.4 Configuring a performance schedule | 227 |
| 8.5 Monitoring using IBM Tivoli Monitoring | 229 |
| 8.5.1 Components of IBM Tivoli Monitoring | 229 |
| 8.5.2 Monitoring agents | 231 |
| 8.5.3 Data collection | 232 |
| 8.5.4 Configuring a performance schedule | 238 |
| 8.5.5 Profiling for real-time observation | 241 |
| 8.5.6 Importing data from IBM Tivoli Monitoring | 243 |
| 8.6 Customizing reports and overlaying counters | 248 |
| 8.7 More information | 253 |
| Chapter 9. Application monitoring | 255 |
| 9.1 Overview of application monitoring | 256 |
| 9.1.1 End-to-end distributed business transactions | 256 |
| 9.1.2 Application response measurement | 258 |
| 9.1.3 Instrumentation | 261 |
| 9.2 Configuring your environment | 264 |
| 9.2.1 IBM WebSphere Application Server | 264 |
| 9.2.2 BEA WebLogic Application Server (WLS) | 267 |
| 9.2.3 Using the Application Server Instrumenter (ASI) | 269 |
| 9.3 Enabling real-time application monitoring | 279 |
| 9.3.1 Architecture | 279 |
| 9.3.2 Configuring Performance Tester | 285 |
| 9.3.3 Profiling for real-time observation | 288 |

| | | |
|--|--|-----|
| 9.3.4 | Finding the causes of performance problems | 294 |
| 9.3.5 | Profiling for real-time observation analysis | 302 |
| 9.4 | Collecting application monitoring data from IBM Tivoli family of products | 306 |
| 9.4.1 | Architecture | 307 |
| 9.4.2 | Importing application monitoring data | 308 |
| 9.5 | More information | 317 |
| Part 2. | Applying RPT to enterprise application testing | 319 |
| Chapter 10. Scalability and test agents | | 321 |
| 10.1 | Introduction | 322 |
| 10.1.1 | Abbreviations | 322 |
| 10.1.2 | RPT process architecture | 323 |
| 10.1.3 | RPT workbench architecture | 324 |
| 10.1.4 | RPT driver architecture | 325 |
| 10.1.5 | Java memory management considerations | 326 |
| 10.1.6 | Measuring memory – getting on the same page! | 328 |
| 10.1.7 | Methodology for determining the incremental virtual user memory footprint | 329 |
| 10.1.8 | Possible reasons for overestimating RPT memory usage | 332 |
| 10.1.9 | Measuring CPU usage | 333 |
| 10.1.10 | Methodology for determining the incremental virtual user CPU utilization | 335 |
| 10.2 | Driver measurements | 337 |
| 10.2.1 | Machine configurations | 337 |
| 10.2.2 | Workload descriptions | 338 |
| 10.2.3 | Memory footprint measurements | 339 |
| 10.2.4 | CPU usage measurements | 347 |
| 10.3 | RPT sizing guidelines | 351 |
| 10.3.1 | Overview and approach | 351 |
| 10.3.2 | System configuration | 353 |
| 10.3.3 | Workbench sizing | 354 |
| 10.3.4 | Driver sizing | 355 |
| 10.4 | Setting driver maximum JVM heap size | 364 |
| 10.4.1 | Remote driver | 364 |
| 10.4.2 | Local driver | 365 |
| 10.5 | RPT 6.1.2 HTTP memory footprint improvements compared to RPT 6.1.1 footprint | 366 |
| 10.5.1 | HTTP memory usage characteristics of RPT 6.1.1 (and 6.1) | 367 |
| 10.5.2 | HTTP memory usage characteristics of RPT 6.1.2 | 367 |
| 10.5.3 | Comparison of Plants41k driver memory usage for RPT 6.1.2 versus RPT 6.1.1 | 367 |
| 10.5.4 | Memory reduction of RPT 6.1.2 vs. 6.1.1 for three workloads | 368 |

| | |
|---|------------|
| 10.6 Intermediate results for memory footprint measurements | 369 |
| 10.6.1 TradeSVTApp | 369 |
| 10.6.2 InternetShoppingCart | 375 |
| Chapter 11. Testing SAP enterprise applications | 383 |
| 11.1 Introduction | 384 |
| 11.2 Testing the SAP GUI for Windows | 385 |
| 11.2.1 SAP GUI for Windows prerequisites | 385 |
| 11.2.2 Recording SAP GUI tests | 389 |
| 11.2.3 Editing SAP GUI tests | 394 |
| 11.2.4 Executing SAP GUI tests | 396 |
| 11.2.5 Evaluating SAP GUI test runs | 397 |
| 11.2.6 Creating complex SAP GUI tests | 397 |
| 11.2.7 Tuning the operating system | 399 |
| 11.2.8 Tips and tricks | 399 |
| Chapter 12. Testing Citrix application environments | 401 |
| 12.1 Introduction | 402 |
| 12.2 Citrix Windows environment | 402 |
| 12.3 Testing Citrix for Windows | 403 |
| 12.3.1 Limitations | 403 |
| 12.3.2 Important guidelines | 404 |
| 12.4 Recording Citrix tests | 405 |
| 12.4.1 Prerequisites | 405 |
| 12.4.2 Recording a Citrix test script | 406 |
| 12.5 Editing Citrix tests | 414 |
| 12.5.1 Test editor | 414 |
| 12.5.2 Test element details | 416 |
| 12.5.3 Citrix session | 417 |
| 12.5.4 Window events | 417 |
| 12.5.5 Screen capture | 419 |
| 12.5.6 Image synchronization | 420 |
| 12.6 Evaluating Citrix test runs | 420 |
| 12.6.1 Citrix performance report | 420 |
| 12.6.2 Citrix verification point report | 422 |
| 12.7 Creating complex Citrix tests | 424 |
| 12.8 Optimizing Citrix tests | 425 |
| Chapter 13. Testing with custom Java code | 429 |
| 13.1 What is custom code? | 430 |
| 13.2 Custom code usage | 430 |
| 13.3 The basics of custom code | 432 |
| 13.3.1 Adding custom code | 432 |
| 13.3.2 An example of custom code use | 435 |

| | |
|---|------------|
| 13.4 Using ITestExecutionServices | 437 |
| 13.4.1 Logging support | 437 |
| 13.4.2 Data areas | 443 |
| 13.4.3 Loop control | 445 |
| 13.4.4 ITransaction | 445 |
| 13.4.5 Using other Java APIs | 448 |
| 13.5 Using datapools | 452 |
| 13.6 Extending tests with custom code | 457 |
| 13.6.1 Passing data to custom code | 457 |
| 13.6.2 Returning data from custom code | 459 |
| 13.6.3 Test duration control example | 461 |
| 13.7 Synthetic workload simulation | 463 |
| 13.7.1 Prerequisites | 463 |
| 13.7.2 Recommended practices | 464 |
| 13.7.3 Application under test state management | 464 |
| Chapter 14. Smart Bank: An example of dynamic workload implementation with RTP | 471 |
| 14.1 Smart Bank: What is it? | 472 |
| 14.2 The demonstration proof-points: A day in the life of a bank | 473 |
| 14.3 Workload analysis | 479 |
| 14.3.1 Particular points to consider | 483 |
| 14.4 Smart Bank showcase implementation | 484 |
| 14.4.1 RunProperties custom code | 487 |
| 14.4.2 ReadProperties custom code | 490 |
| 14.4.3 TestName custom code | 492 |
| 14.4.4 Account custom code | 493 |
| 14.4.5 RunState custom code | 494 |
| 14.4.6 DoThinkTime custom code | 495 |
| 14.4.7 Test personalization | 496 |
| 14.5 RPT V7 architecture for workload injection | 497 |
| 14.6 Conclusion | 498 |
| Part 3. Appendixes | 501 |
| Appendix A. Installation and licensing | 503 |
| Product installation | 504 |
| Installing the Installation Manager | 505 |
| Installing Rational Performance Tester | 509 |
| Installing Rational License Server | 518 |
| Installing Rational Performance Tester Agent | 523 |
| Product licensing | 530 |
| Product activation kits | 531 |
| Product playback licensing | 531 |

| | |
|-----------------------------------|-----|
| Abbreviations and acronyms | 533 |
| Related publications | 535 |
| IBM Redbooks | 535 |
| Other publications | 535 |
| Online resources | 536 |
| How to get IBM Redbooks | 536 |
| Help from IBM | 537 |
| Index | 539 |

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

| | | |
|-----------------|---|----------------------------|
| AIX® | eServer™ | System z™ |
| Candle® | Extreme Blue™ | Tivoli® |
| CICS® | IBM® | Tivoli Enterprise Console® |
| ClearCase® | OMEGAMON® | WebSphere® |
| ClearQuest® | Rational® | xSeries® |
| DB2® | Redbooks® | z/OS® |
| developerWorks® | Redbooks (logo)  | z9™ |

The following terms are trademarks of other companies:

SAP NetWeaver, SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Adobe, Acrobat, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

EJB, Java, JavaScript, JDBC, JDK, JNI, JVM, J2EE, J2ME, J2SE, Portmapper, RSM, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ActiveX, Excel, ESP, Internet Explorer, Microsoft, MSDN, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication is intended to show customers how Rational® processes and products support and enable effective systems testing.

The book describes how performance testing fits into the overall process of building enterprise information systems. We discuss the value of performance testing and its benefits in ensuring the availability, robustness, and responsiveness of your information systems that fill critical roles for your enterprise.

Based on years of project experience, we describe the key requirements needed in performance testing tools and how the IBM Rational Performance Tester tool was developed to meet those requirements. We also walk through the choices that we made to steer the tool architecture into using an open source platform as its base with Java™ as its language to permit ubiquitous platform support.

This book is structured into two parts:

- ▶ Part 1: Understanding Rational Performance Tester
- ▶ Part 2: Applying Rational Performance Tester to enterprise application testing

This book is not an exhaustive reference manual for the Rational Performance Tester product. The product has reference material built-in to the help system and can be accessed through the Help menu or using the F1 key to get context sensitive help for the current user interface item with focus.

Target audience

The target audience for this book includes:

- ▶ Managers, system architects, systems engineers, and development staff responsible for designing, building, and validating the operation of enterprise systems
- ▶ Rational and other software brand services and sales representatives and key partners who have to understand and utilize Rational Performance Tester
- ▶ Staff members who have to use Rational Performance Tester to size hardware systems, and consultants who have to validate system operation under projected production conditions

The team that wrote this book

This book was produced by a team of specialists from around the world working remotely under the guidance of the International Technical Support Organization, Raleigh Center.

David Chadwick is the Level 3 Technical Support Manager in the IBM Rational Performance Tester development organization in Raleigh. He also serves as the product champion and SWAT team member responsible to help internal organizations and customers adopt the Performance Tester tool and associated performance testing processes. He has spent over 20 years working in the areas of computer performance and related tool development. This is his first IBM Redbooks project, which he conceived and gathered an all-star cast to help him write. David holds a Masters of Science in Electrical Engineering specializing in Computer Engineering from Stanford University and two Bachelor of Science degrees from North Carolina State University in Electrical Engineering and Computer Science.

Chip Davis is the Modular Service Offering (MSO) development lead for Rational Services in Atlanta, Georgia. He has been involved in providing pre-sales and post-sales services for over 6 years and was an early adopter, training material developer, and trainer for advanced users of the Performance Tester product. Chip has a Bachelors of Electrical Engineering from Georgia Institute of Technology.

Mark Dunn is a Software Engineer on the Performance Tester development team in Raleigh and also an on-going contributor to the Eclipse Test and Performance Tools Platform project. He works on the network proxy recording technology. Prior projects at IBM include the WebSphere® Workbench Simulator project, which was another performance testing tool.

Ernest Jessee is a Software Engineer on the Performance Tester development team in Raleigh who specializes in statistical analysis tools and techniques, statistical data aggregation, and Performance Tester workbench performance. He has worked on performance testing tools for over 7 years.

Andre Kofaldt is a Senior IT Specialist in Rational Services in Köln, Germany. He was an early adopter of the Performance Tester product and has provided services to several IBM customers using the tool. He was also one of the first users of the SAP® extension.

Kevin Mooney is a Software Engineer on the Performance Tester development team in Raleigh and specializes on the tool's playback engine performance characterization and tuning. He has worked on performance testing tools for over 10 years.

Robert Nicolas is an IT Specialist working on the System z™ Benchmarking team at the Products and Solutions Support Center in Montpellier, France. He has extensive experience using many performance testing technologies to provide system sizing information to IBM's customers.

Ashish Patel is a Software Architect and Development Lead working at the IBM Toronto Software Lab in Canada. He has over 7 years of industry experience in software architecture and development and 4 years of business development and entrepreneurial experience. Ashish participated in IBM's premiere internship program, Extreme Blue™, where he co-founded the IBM Performance Optimization Toolkit (IPOT). He has worked with IBM for 2 years on numerous start-up products in the software development and test area. Prior to joining IBM, he created software solutions for one of the largest integrated petroleum companies in the world and one of Canada's top providers of energy. He also founded a privately-held software development and consultancy firm, where he was appointed President and Chairman of the corporation, and which he operated for 4 years. Ashish holds a Computer Engineering degree from the University of Alberta.

Jack Reinstrom is the System Performance Test Manager for the System Verification Testing team for the IBM Rational server-based tools in Lexington, Massachusetts. He and his team have adopted Performance Tester to perform all of their testing. Jack has led the effort to use custom Java coding to drive their tool APIs when not using standard client-server protocols.

Kent Siefkes is the Second Line Development Manager for the Performance Tester development team in Raleigh. He has been architecting and managing the development of performance testing tools for over 20 years.

Pat Silva is an Information Developer for the Performance Tester development team in Raleigh. She has developed performance testing product documentation for 10 years.

Susann Ulrich is a Program Manager for Quality Management in the TechWorks team in Rational Sales in Miami, Florida. She has worked for Rational for over 10 years. Susann has been instrumental in the development and delivery of technical enablement for Performance Tester to the Rational sales team.

Winnie Yeung is an IT Specialist in IBM Global Business Services in Toronto, Canada. She was one of the first users of the Citrix extension for Performance Tester.

Ueli Wahli of the International Technical Support Organization, Raleigh Center, has been a Consultant IT Specialist at the ITSO for over 20 years. Ueli helped with the writing tools and initial editing of this book.

Thanks to the following people for their contributions to this project:

- ▶ **Marty Swafford**, Senior Product Manager, Rational Curriculum Development, Lexington, Massachusetts, for help with managing the project and production of book drafts for review.
- ▶ **Anuradha Ramamoorthy** is a system performance test engineer on the System Verification Test team for several IBM Rational server-based products. She has been a pioneer and principal contributor in the testing of Rational products using custom Java code with Performance Tester and has been a performance test practitioner for over four years.

Thanks to those who provided technical review comments to the initial draft: Allan Wagner, Barry Graham, David Hodges, Fariz Saracevic, Karlene Vilme, and Tom Arman; also, thanks to Yvonne Lyon, for editing this book

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an e-mail to:
redbooks@us.ibm.com

- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Part 1

Understanding Rational Performance Tester

The theme for this part of the book is to explain:

- ▶ **How does this feature or function work in RPT?**

We cover the basic usage flows for how the tool user goes from recording a user scenario to test execution and then performing test results analysis.



Introduction

This chapter starts by describing how performance testing fits into the overall process of building enterprise information systems. We discuss the value of performance testing and its benefits in ensuring the availability, robustness, and responsiveness of your information systems that fill critical roles for your enterprise. From this premise, we develop the requirements for a performance testing tool that can aid in filling this value proposition.

These requirements from the problem space are then combined with some basic business and marketing requirements to lead us to a product architecture for the IBM Rational Performance Tester tool. We also walk through the some of the choices that we made including development on top of an open source platform and choosing Java as its programming language to permit ubiquitous platform support.

1.1 Overview of IBM Rational Performance Tester

When developing the requirements for a new testing tool, we first took a look at the business and technical problem space. From there we took a harder look at the customer, including the non-tool oriented skills needed to analyze its results along with the history of performance testing tools that are in current use in the market. With this work as background, we formulated design goals for the new tool that would provide a quantum leap of improvement in productivity, scalability, ease, and depth of analysis, and extensibility for new environments.

1.1.1 The performance testing problem space

Many people associated with a company are impacted by the performance of its enterprise computer systems. Companies run on computer systems from ordering to billing to payroll systems. What are the consequences of these critical systems having poor system performance? Let us examine the following three examples of critical business systems and what happens when they do not respond as expected:

- ▶ The company Web site is the face of the company on the Internet. How does it reflect on the company if someone should search for the company name, find its Web site, and get an Error 404: File not found when clicking on the home page? The user goes back and looks up another vendor whose Web site is currently working properly.
- ▶ A sales specialist is on the telephone with a new customer who is interested in the flagship product and has asked for pricing for 1000 units. The customer is kept waiting for several minutes. The specialist then has to apologize and promise to call back because the order system rejects his login and password because it is overloaded and cannot create his user connection.
- ▶ The division sales manager tries to run an ad hoc report on the sales projections and closures five days before the end of the quarter with a breakdown by salesperson. She wants to use the results in her weekly 4pm staff meeting so she can encourage those with big projected deals that have not yet closed and enlist help for those that need it. The heavily burdened and poorly tuned sales system responds so slowly that the report request submitted at 3pm finishes after 5pm. She is forced to call a second special staff meeting the next business day to cover the report, and the resulting lost time leads to several deals slipping out of the quarter.

You might think that these examples are extreme or contrived, but in fact, these cases are typical and happen in every industry. Having your enterprise systems operating at peak efficiency with ample capacity is a strategic advantage and in some cases is a bare minimum requirement to compete in some market spaces.

Let us examine another very common scenario that occurs frequently in many companies facing the challenges of a global economy and the impact of competition. The XYZ Widget company has made a strategic decision to move its manufacturing to a plant in the Pacific basin to save on labor costs and to import some raw materials from South America used in the widget manufacturing process. This directive sought by the CEO and approved with overwhelming support by the board of directors must be implemented in the next six months or the company risks badly missing its profit targets for the fiscal year. The move of the manufacturing plant has gone well and beyond expectations because a partner was found with capacity to implement a new widget manufacturing line in only three months.

However, the IT shop is struggling to add the significant changes required to the ordering, inventory, and distribution management system. They are forced to bring in consultants to radically change their system and are getting ready to put the modified system into production. At the end of the month, the new system will be cutover into production, the existing records will be updated to the new formats, and all of the new capabilities to control and track the orders and inventory across continents will be activated. There will be no going back to the previous system. If the new system fails, the company's operation will grind to a halt. What is it worth to test this new system under production conditions before the cutover?

1.1.2 Need for a performance testing tool

How do you implement a performance test of an enterprise computer system? It is clear that providing the assurance of a successful production cutover is very valuable to every company. Do you ask your entire sales force to come in on a Saturday to login to the new system and try to simulate their normal daily activities and see if the system can handle it? What if the new system proves to be poorly tuned and needs some adjustments or even limited redevelopment? Do you ask them to come in for the next four weekends while you get the system in shape for cutover? What does that do to morale and productivity? What you need is a laboratory based tool to simulate the workload and measure the system's performance.

There have been performance testing tools available in the general marketplace since the advent of minicomputers in the early 1980s. Before that, there were proprietary tools developed by computer vendors for internal purposes back as far as the early 1970s and maybe even earlier. However, in today's environment every company has a need to test their enterprise systems before putting them into production use. Equipping and staffing a system performance testing laboratory is now an essential stage in enterprise system development for any IT shop. Purchasing a performance testing tool capable of full capacity testing of your production system is a critical asset in the performance test lab.

1.1.3 Goals of IBM Rational Performance Tester

In this section, we have listed the primary design goals we used in building the IBM Rational Performance Tester product. Each goal stands as an independent choice, but as a group, they drove the tool architecture into the uncharted waters of open source platform development using a non-real time programming environment. The technology challenges introduced by this direction were daunting and took a team of seasoned performance tool builders into new development modes replete with use case oriented requirements, subsystem and feature designs using object oriented modeling, automated unit testing as part of the daily product builds, and repeated benchmarking of the tool's capacity to drive larger workloads with fewer system resources.

The key for any tool development is to start with clear goals and tie every decision and technology choice back to those goals. These goals drove many product decisions from ordering which features came first, to deciding how the tool generated artifacts were stored and persisted.

Start with the Web-based testing space

A large portion of newly developed enterprise systems are based on Internet connectivity of a geographically distributed work force that all need on-line access to the system. Web browser and server technologies have proved to be the superior technology for several important reasons: Vendor independence, heterogeneous system support, Internet-based connectivity, and ease of connectivity with legacy systems. As we survey the current trends in the marketplace, more than 80% of the requested enterprise performance tests are using Web browser connections. Therefore, having a solution for Web-based systems is the best place to enter the performance testing tool space with a new tool.

High productivity with no programming

In many companies, performance testing is not done by a specialist but rather by someone with other responsibilities such as the product architect or a business analyst. To broaden its usefulness, a performance testing tool should not require knowledge of a proprietary programming language, or even a special library functions in a standard programming language. Basic skills required to use a performance testing tool should only include knowledge of how to schedule and scale the user scenarios and where to put in the data variation for the application being tested.

High extensibility with Java coding

There are many Web-based applications that take advantage of the browser's ability to run Java applet code to do sophisticated operations locally. Many times the user scenarios require these applet operations to be simulated in some limited way so that subsequent requests to the server contain the correctly computed dynamic data. This type of custom coding should be supported in a well known standard language that is platform independent and widely available. Java makes an ideal choice for any tool's extensibility language.

Customized reporting with all classes of data

The whole point of measuring application performance in performance testing is to analyze the test results, find why the system responds the way it does and tune it if necessary. Over the past decade with the explosion of Web-based applications, the number of users simultaneously accessing these enterprise systems has risen dramatically. The idea of reporting complete and insightful test results has become a challenge because of wealth of data points that need to be reduced statistically to provide an accurate view of these complex systems.

Now that the number of data points has moved from 100,000 to 100 million individual measurements, a design imperative is to reduce the data in real time during the test in a distributed fashion. This principle is coupled with the increased complexity of the tested system architectures yield a need to have complete flexibility in reporting and correlating a variety of data sources. Most of these analyses are dependent on the user's system architecture and therefore must be custom designed to show the results needed for that system.

1.2 Architecture of IBM Rational Performance Tester

Based on the design goals, the product design team who has been designing for this market space for almost twenty years put together a robust tool architecture. Having experience in solving the basic problems of a scalable performance testing tool, some design decisions were oriented to remove the limitations of previous generations of tools. These include platform independence so that the tool can have a single platform build process; distributed data collection, reduction, and display in near real time during the test; a higher level visual abstract test representation that can be used by non-programmers and programmers alike; and a highly scalable playback engine that does not increase operating system process or thread usage with the addition of more virtual user sessions. In each case these design decisions were made to extend the tool's capabilities beyond the current generation of tools.

1.2.1 Eclipse and Java

To increase the general appeal of the tool, the team decided to build it on a set of industry standards and open systems environments. This strategy permits the use of established technologies that are already present in the marketplace and provide a distinct alternative to the proprietary designs of the dominant tools in the marketplace.

By capitalizing on the Eclipse user interface technology, people familiar with it would be instantly more productive. Also you have the immediate advantage of progressive refinement and improvement as Eclipse evolves. The entire infrastructure of open source developers, publications, training, and industry conferences that surrounds Eclipse can be leveraged to gain acceptance for the tool.

By choosing Eclipse, there is also the advantage of being supported on a large number of platforms including Windows®, UNIX, and Linux®. It can be written and maintained as a single code base and run in any of these environments.

Java, as the programming language, is an easy choice. It is a platform independent, freely available, and very popular language that is taught as the first programming environment in the overwhelming majority of colleges and universities. Although not in the mainstream of most testing shops currently, there certainly is ample industry availability of Java savvy resources.

1.2.2 TPTP as a tool framework

Because this is a new entry into the performance test tools market and there are several existing solutions out there, the tool must climb the functionality curve very quickly to catch and surpass the other tools available. The only way this is possible in today's software development economy is to create a rich extensible environment and give others the ability to participate in improving the solution. There was a strategic decision to build a testing tool platform and give it away to eclipse.org as an open source project named Eclipse Test & Performance Tools Platform (TPTP). With this in place, other vendors can build on the same platform and help improve the platform.

Also, there has to be a way where business partners and independent software vendors can add incremental functionality to the basic tool as developed by the core development team. This provides other internal business units that have specific needs with the ability to extend the tool and take advantage of the core product functionality. If they had to wait for the tool development team to build a solution specific to their needs, it might not be built or at least not soon enough.

The tool has included an extensibility interface for this purpose. Several additional application specific solutions have already been implemented using this interface. It has proved to be of great value to encourage the formation of an environment of innovation and extensibility around the tool. If a development organization decides to choose the tool as the basis for all of their performance testing, it can extend the tool to meet all of its project specific needs.

1.2.3 Building a complete tool chain

A productive performance testing tool takes advantage of the normal operation of the application to predict the expected responses that might be encountered during a large scale performance test. A good tool must be able to capture the client/server interactions and automatically generate a test that is ready to playback without changes. The test playback process had to proceed after very few user actions, such as a tool bar button or a right-click menu option on the test to be played back. Visual indication of the progress of the test playback must be in evidence without any user action. Test results should be automatically generated and stored in a logical place and timestamped.

The tool has a capture, edit, replay, and analyze usage pattern. Many of the underlying transformational steps in the process are executed automatically without user intervention. All steps of the testing process should be done automatically when a change in an earlier part of the tool chain is recognized. For example, when an edit is made to the test, it will be automatically consistency checked on saving. Then when the test is first to be executed, the test will be automatically transformed into Java code that is compiled, archived, and shipped to the test agent for execution along with any support data files.

1.2.4 Recording in a distributed environment

Today's multi-tiered and distributed enterprise systems normally can be accessed from a Web browser at a single client personal computer. For these systems, recording can be done directly on the client personal computer, which is also the host for the tool software. In certain circumstances, recording the traffic between an application server and a back-end server hosting a Web service interface might be wanted. The tool architecture permits *redirecting* remote sources at the recorder to enable testing such as server-to-server applications. In fact, any application that is capable of communicating through HTTP or SOCKS proxy server can have its Web traffic recorded. The tool then does extensive analysis of the recorded traffic and builds a sophisticated test, complete with automatic dynamic data correlation.

1.2.5 Scalable distributed playback

With the adoption of large scale corporate intranet-based infrastructure systems, large corporations heavily depend on these systems to conduct their daily business. It is crucial that any changes to these systems do not disrupt their business. Therefore, changes to the hardware or software of these systems that support anywhere from a few hundred to tens of thousands of simultaneous users must be verified before they are put into production. This tool must therefore support these very large scale user requirements.

Developing a test agent based playback architecture where each test agent runs independently of the other test agents is very powerful. The scalability of the playback complex is completely linear with the number of test agents employed, because there is no real time coupling between test agents, or between the test agent and the workbench. The resulting data is collected locally on each test agent during the run, summarized into a statistical periodic update packet, and sent back to the workbench to provide near real time data to the tool user looking at the workbench.

After the test run completes, any additional detailed test log data is transferred back only after the performance test data has been stored in the project. If the test log data is too large for the workbench to handle, the performance test run results still remain available and are completely valid. By being able to adjust the period of the statistically update, the load on the workbench from dozens of test agents can be reduced to a manageable level.

1.2.6 Test run monitoring and data collection

Previous generations of performance tools collected all of the response time data on the test agent, and then sent all of the low level data back to the console for processing after the test run is over. Therefore, the user of the tool got no real time performance feedback during the test run. This was seen as big deficiency and caused many test runs to be run to completion and then thrown out when the test results indicated that the system or the load simulation was not working properly.

In addition, you might want to take test readings from multiple system load levels as a part of a single test run. Unless you can sense a steady state interval interactively during the test, you do not know when to move to the next load level. For these test interval efficiencies and the ability to get instant feedback on the performance results, the tool has been designed to provide complete test results, including resource monitoring data, response times, and transaction throughputs to show in near real time during the test run.

1.2.7 Post run analysis

Certain additional analyses are more easily done after the test has completed; for example, scanning the overall results to make sure that virtually all of the verification points passed during your test run. Other metrics can help determine that the response time data is valid. You need enough data points to prove that the response time average is not time correlated, is normally distributed, and containing enough data to trust the results. When you know all of this from the response time versus run time graph and the standard deviations across the steady state intervals, you can establish your test results as valid statistically.

The other analysis that has to be done typically is to find the bottleneck or slow components of a complex system. There are two sets of measurements that help determine where the system is limited. The first class of measurements is computer resource utilization. These hardware and software metrics permit you to understand the overall utilization of hardware and operating system resources. When the major bottlenecks from the system have been removed, and several of the subsystems are operating at maximum capacity, you have an efficiently tuned and balanced system.

The other set of measurements is the breakdown of all the critical transactions of your application into their constituent parts. Finding those segments of time spent that are the largest components of the overall response times yield the bottlenecks of the system. These are the areas where system tuning and application redesign can improve the overall system performance.

In today's systems, the performance testing process requires expensive hardware and software resources as well as a well trained staff of experts. Tools should reduce the complexity of preparing for the measurement interval or intervals (typically). In addition if the testing interval can be compressed by running many of the load levels required in a single test session, the overall time it takes to get your test results is reduced. Supporting multiple load levels based on elapsed time intervals and observing the beginning and ending of each set of steady state measurement intervals permit reporting of measurements for each interval independently. This is a highly productive approach encouraged by this tool and is now fully supported.



Performance testing process

This chapter describes the basic process of implementing a performance testing capability within your organization. You must make several investments beyond just purchasing a performance testing tool to be successful and create a sustainable performance testing capability.

In this chapter we describe the performance testing process and the artifacts that are generated as you complete each of the process steps. Finally, we provide a set of example artifacts that, taken together, provide a basis for understanding how the performance testing process can be carried out.

2.1 Establishing a performance testing capability

Most reasonably sized IT organizations must establish a performance testing capability as one of their standard functions. Taken as an organizational function, we discuss the constituent parts of building this capability. Beyond the obvious organizational elements of budget and staff, you need *SWEAT*: Steps, Workload, Equipment, Analysis, and Tools.

- ▶ **Steps** refers to the need to follow a standard performance testing process. By obeying all of the important stages in this process, you can be assured of a reliable conclusion from your performance test.
- ▶ **Workload** refers to the user scenarios modeled, rates of execution, and the identified measurements used in the performance test. Some performance tests can yield significant results and system improvement by just running a stress test without an attempt to model the anticipated production workload on the system. In other cases where you are trying to ensure that the system can handle its intended workload, you should perform the business analysis or distill it statistically from the current production system if possible.
- ▶ **Equipment** refers to the laboratory environment necessary to implement a full scale performance test. Often the system under test is undersized or the hardware required to drive the test is not anticipated.
- ▶ **Analysis** refers to the need to assemble a multi-disciplinary team including the system and application architects to participate in performance testing. They are instrumental in the tuning and subsystem capacity testing as well as during the final full scale testing process. Without their input, the system tuning would have to be done superficially without the intimate architectural knowledge of possible optimizations and is less likely to result in an acceptable final result.
- ▶ **Tools** refers to the adoption of highly scalable and productive testing and analysis tools that work with the latest generation of enterprise servers and their middleware. Today's toolset should include flexible and dynamic load generation, response time measurement, operating system monitoring, and distributed application analysis capabilities. Without these functions, today's highly complex enterprise applications can be difficult to analyze and tune.

Successful performance testing requires a long term commitment by the organization. Any enterprise that runs its business on a software rich infrastructure cannot only survive, but adapt and thrive by evolving its infrastructure as business conditions change. Having a capability of repeatedly performance testing new versions of their software infrastructure is like an insurance policy against introduction of poor quality software modifications. This information technology infrastructure is a strategic asset to businesses competing in today's global economy.

2.2 The performance testing process

Performance testing is a well understood discipline that has been practiced for well over thirty years. First there were time sharing capabilities on mainframe computers, then minicomputers with multiple asynchronous terminals, and later networks of personal computers connected to server systems. Testing all of these systems revolved around the need to understand the capacity of the shared portions of the system.

The process of performance testing has not changed significantly since these earlier system types were being tested. However, the complexities of the system design with more distributed intelligent hardware components, and many more interconnected software subsystems yield more challenges in the analysis and tuning parts of the process. On current systems, performance testing should be done iteratively and often during the system design and implementation. Tests should be performed during implementation of critical subsystems, during their integration together into a complete system, and finally under full capacity workloads before being deployed into production.

The basic steps in the performance testing process are listed here. Each step is explained in the sections that follow in terms of the actions taken and results of the step:

- ▶ Determine the system performance questions you wish to answer.
- ▶ Characterize the workload that you wish to apply to the system.
- ▶ Identify the important measurements to make within the applied workload.
- ▶ Establish success criteria for the measurements to be taken.
- ▶ Design the modeled workload, including elements of variation.
- ▶ Build the modeled workload elements and validate each at the various stages of development (single, looped, parallel, and loaded execution modes).
- ▶ Construct workload definitions for each of the experiment load levels to collect your workload measurements.
- ▶ Run the test and monitor the system activities to make sure that the test is running properly.
- ▶ Analyze measurement results and perform system tuning activities as necessary to improve the results and repeat test runs as necessary.
- ▶ Publish the analysis of the measurements to answer the established system performance questions.

2.2.1 Performance test goals

One of the most critical factors in the success of a performance test is goal setting. As the saying goes, if you are not aiming at anything in particular, then you will surely hit it. Performance tests tend to have many stakeholders that might not be obvious at the beginning of the project. You should ask the direct recipient of the project results who else will be reviewing and approving actions based on the results of the performance test. Enumerate these additional stakeholders and make sure their expectations are taken into account in the list of test goals. There has been more than one performance test project that had to be restarted or had its results rejected because it did not meet the expectations of someone who was not involved in the formulation of the original project goals.

There are some fairly standard goals for performance tests that you can use to start from to elicit a solid direction for the performance testing project. Often several of these are the expected result by various stakeholders. You have to explicitly state the goals that you are working towards as a way to set both positive expectations of what goals you are planning to achieve as well as setting negative expectations of the goals you are not attempting to meet.

Examples of performance test goals include:

- ▶ Stress testing the application to prove robustness
- ▶ Measuring the overall system capacity in simultaneous user or transaction rate loading
- ▶ Reliability testing of a given workload over a period of days or weeks to gauge an approximate availability metric or a predicted mean time to failure
- ▶ Response time measurement of critical transactions or user actions during a projection of the production maximum workload
- ▶ Overload testing to characterize the system's responsiveness as the workload exceeds the system's capacity
- ▶ Load range testing with incremental increases in load to show the system's sensitivity to changing load
- ▶ Tuning, balancing, and optimizing the subsystems and ultimately the complete system to provide maximum balanced system capacity for the lowest unit cost per user or transaction

By writing down the performance testing goals (usually as part of the workload specification) and reviewing them with the stakeholders, you can begin your testing project with confidence that you can complete the project with a satisfactory set of results and conclusions. Without this agreement, the pressure and scrutiny of high profile stakeholders can make the project difficult if not impossible to complete successfully.

Performance testing is a very high profile and stressful activity because the decision making based on it can have major business implications. By having everyone in agreement before the test results are reported, there will be less opportunity to second guess the testing goals, approach, workload definition, or chosen metrics.

2.2.2 Workload model of the system's busy hour

The workload model usually consists of a characterization of how much work the system must perform at its busiest time. This represents the peak loaded capacity that the system must support while maintaining satisfactory user visible characteristics such as responsiveness to user requests for information. The derivation of the workload model is an important skill expected of a senior systems analyst, a system architect, a business analyst, or a performance testing specialist. However, without a clear analytical workload model, the performance testing project cannot truly get started. Often the performance tester is expected to bring together the business analyst and system architect and build a workload model that describes the busy hour (or hours) of the developed system.

The workload model might be derived from the number of business transactions expected during the month-end or quarter-end rush. It might also be based on the late afternoon batch report run interval that overlays an otherwise uniform transactional workload. It could also be at lunch hour on the Friday after the Thanksgiving holiday for a seasonal e-commerce Web site.

In most cases, you can think of a workload model as a matrix of transactions (or business scenarios) versus frequency of execution spread across the number of users accessing the system simultaneously. There are many forms used to describe this quantity of work. One example is shown in Table 2-1, where the simplified version of the system's load is reduced to two types of user transactions and two batch reports that are run in the busy hour.

Table 2-1 Example of a workload model

| Business transaction | System-wide total / hour | Simultaneous users | Average think time [seconds] | Records impacted |
|----------------------|--------------------------|--------------------|------------------------------|------------------|
| Browse item | 50,000 | 5,000 | 3.00 | 1 |
| Buy items | 500 | 100 | 5.00 | 3-5 |
| Inventory report | 1 | 1 | 0 | 200,000 |
| Sales report | 1 | 1 | 0 | 8,000 |

The impact on the system across all four components is measured and analyzed. Depending on the system design, any of the four components could cause significant interactive or batch reporting degradation beyond the minimum acceptable level of performance expected.

2.2.3 Measurements and metrics

Normally there is a collaboration between the business analyst and the system architect that serves as the source for the performance requirements for the system being built and deployed. Ideally these requirements were stated at the time of the initial system design and only need to be qualified and clarified in measurable terms so the performance testing project can validate that the system meets its stated performance requirements. However, based on plenty of project experience, this situation turns out to be less common than you would expect. Many enterprise system projects get commissioned to solve a particular business need without any thought or concern about the expected performance of the system. In these very common cases, the precise statement of the performance requirements is left as an exercise to the performance tester.

When presented with a performance testing project without written performance requirements, the tester can and should broker the setting of these requirements. The performance tester gathers these requirements by whatever means are at their disposal. Hopefully, this includes access to the business analyst and the system architect who have some knowledge of the business environment of the deployed system. The tester then captures the performance requirements in a measurable form and reviews them with the project stakeholders to gain their agreement.

It is very important that the tester does not set the performance requirements in isolation, make system measurements, and present a set of performance results that recommends slipping the deployment date or a total system redesign. This is the ultimate recipe for a disastrous project result. The requirements review has to precede any system performance measurement to ensure their acceptance. Having to rejustify the workload definition, the measurement points, or the accuracy of the tool software are among the project distractions that can derail a performance test project. The performance requirements should be agreed to and socialized properly among the stakeholders before the measurement data is available. This will keep the focus of the project on tuning the system and making sure the system performs at an acceptable level.

2.2.4 Test success criteria

Ideally the workload model will describe—with a fairly (80-90%) accurate mixture of the transactions (or business workflows)—the expected peak production workload on the system when it is in use by the business. By liberal application of the 80/20 rule, the workload model should contain the 20% of the transactions that make up 80% of the system capacity, and contain most if not all of the crucial business transactions.

Measurement of the throughput of the required transactions is likely to be a minimum standard that must be met by the system. For example, if during the peak Christmas rush, an online ordering system must process 500 shipments per hour to meet the projected increase in order volume, based on the previous three years statistics, then the new system must support this transaction rate.

On the other hand, if the submit order and order completion screens each take more than 5 seconds to appear, then most customers will abandon their shopping cart without purchasing, cancel their orders, or not return for subsequent purchases. These are more customer-oriented requirements that in fact dictate the success of the online shopping site system deployment.

Both types of requirements must be measured and successfully achieved or there is a significant business risk in going ahead with system deployment. By helping the system development team focus on these metrics and the underlying causes of non-compliant transactions or response times, the performance tester can drive a successful outcome of the performance testing project.

2.2.5 User scenarios in the workload model

Business workflows are also considered to be the user scenarios of the system when viewed from the user's perspective. By walking through the typical user actions that make up each of these business workflows, you can create a user scenario (also known as a use case in the software requirements world) complete with typical user inputs such as login, password, product selections for a commerce site or an account number, transaction type, and amount for a financial site.

The number of user scenarios that make up the workload model is very subjective and can have a major impact on how long the performance testing project takes. Simplifying the user scenarios and reducing their number have many benefits. Many times a revision of the application requires that all test scripts be recaptured. This can occur because the workflow changes in the sequencing of user screens, user input data, or even invisible underlying application parameter data passed back and forth between browser and server.

What might seem like a trivial application change from a user flow perspective can cause the test scripts to stop working properly. If you have only ten user scenarios and a highly productive tool such as Performance Tester, an application version change can be accommodated in a half day of recapture, minor editing for input data variation, and test debugging. If you have 20 or more user scenarios, this same event could cost the project a day or more of delay.

Also, the number of alternate flows, conditional execution, and more extensive data variation adds to the overall time to build and maintain the set of tests that implement the workload model. Keeping the user scenarios basic and central to the important business workflows of the system is an important concept to obtain valuable, timely performance results for a reasonable project cost. When you accept the need to abstract the workload definition to a simplified version of the true user behavior (with virtually equivalent system performance), your performance testing projects will proceed more quickly and cost effectively.

2.2.6 Input data variation in user scenarios

Some user inputs, if varied on a per user or per iteration basis, will have an impact on system performance while other user inputs do not. By reviewing the user scenarios with the system architect, the tester can validate the selection of which user inputs to vary. By varying the data values in the user scenarios that represent the independent keys of accessed database tables, representative data retrieval times can be expected. The tester can reduce the possibility of getting optimistic test results due to database or other subsystem caching effects.

In some cases, such as the browsing of an online commerce site, repeatedly selecting the same five items to view in the online catalog might result in the Web server caching images and even product details. However, by adding random selection of the product viewed, the system will experience a much heavier load on the non-Web server subsystems such as an image server or a back-end database containing product details. This difference in the workload might impact order processing times as well as providing a more realistic estimate on the time to view a product detail page. This could be the critical customer requirement that determines the success of the Web site.

2.2.7 Debugging user session tests

For each user scenario, a test recorder is used to capture the user session and generate a test. The first step in debugging the user session is to play this session back as it was recorded to make sure the server accepts the test as generated.

In the event that this does not provide the intended result, it can be caused by four main reasons: an unsupported authentication mechanism is used by the server, the data correlation required by the server application is not automatically handled, a data dependency exists in the user scenario that requires data variation, or some Java applet operations must be programmed to permit the session to run properly. An experienced user of the performance testing tool should be able to handle any of these four cases and tell them apart.

There are four further steps in debugging your test:

- ▶ Add input data variation, verification points, and any flow of control constructs needed for your test when it is run in multi-user mode. Run the test straight through with these changes and make sure that it executes properly and all verification points pass.
- ▶ Create a test schedule and run the same test in a loop of three iterations. Make sure that no test start/end boundary conditions cause failures. Contents of the cookie cache are often a source of trouble here.
- ▶ Create a test schedule and run 3-5 virtual users running the same test in a loop of three iterations. This permits data collisions, table locks, and other simultaneous user errors to be uncovered. Problems here are typically solved by fixing single-threaded application modules and adding data variation where it is required.
- ▶ Create a test schedule and run 20-30% of expected system capacity. With this test, server time-outs and basic load-triggered timing dependencies can be uncovered. In general, these problems are due to application design and tuning deficiencies. The important thing is to uncover these problems early and get application development help in solving them before serious load testing begins.

2.2.8 Workload model for each experiment

In the performance test planning for the project, the tester designs a series of experiments (or test executions) that taken together provides a methodical approach to developing the necessary test assets, making necessary test measurements, permitting the required subsystem and system tuning, making conservative estimates for application changes and test re-runs, and performing the final measurements and their analysis.

Often there is a requirement for building a subset of the full workload model to provide a means of tuning one or more of the subsystems, such as tuning the capacity of a single application server normally used in a clustered environment with a Web server front end. This can be accomplished by only having a single application server active in the cluster or by redirecting the test connections directly to the application server (bypassing the Web server and load balancer).

Problems such as heap fragmentation, memory leaks, and algorithmic inefficiencies can be more easily be addressed in this smaller, less complex environment. Tuning the application server including how many instances to run on a hardware server can be done prior to scaling the system up for a full load test.

Some subsystem tuning and testing involves several subsystems and have to be exercised before the full load tests. One example of this is the database connection pooling to the back-end database. By exercising a heavy load of database intensive operations, possibly without including the entire load of non-database related workload, the database connections from the application servers to the database can be sized properly.

As each of these experiments are defined, an associated workload definition should be listed for that experiment. By pulling these together as part of the test plan, the entire testing project team can understand when the tests and workload definitions are going to be used and what measurements and data collection needs are expected for each experiment.

2.2.9 Test execution results and data collection

Each different experiment has different goals. These goals dictate the size of the workload, its ramp-up period, and what measurements are taken. For example, if the experiment is to discover the full system capacity with the current system configuration and tuning, you can try ramping up one virtual user per second in a balanced manner to say two thirds or three quarters of the system's expected capacity.

When you attain this load level, study the run statistics for a few minutes at that level to make sure all the transactions are still operating correctly. When this has been verified (usually by verification point statistics and steady hit rates), manually add another 25% to the workload and repeat the verification. Add another 25% and perhaps yet another, until you have clearly exceeded some threshold of acceptability. This might be when response times become 2-5 times too long, transaction rates start dropping or staying stable over a wide load range, or the number of failing verification points become unsatisfactory. Using this kind of experiment, the verification point, response times, and transaction rates can be examined to determine the maximum capacity of the system in terms of the number of virtual users.

Another type of experiment is used to delve into why a certain transaction or class of transactions are taking the system too long to process. After knowing the approximate system capacity, you set up a test run for only a few minutes of steady state transactions. This test should be run with about a 50% load, but with the application server instrumentation turned on at the highest level.

In addition, all of the servers (Web, application, database, and any other) key operating system resources should be monitored. When you have captured the data for the test, you terminate the test after no more than 10-15 minutes of steady state data. Doing this should ensure that the data is not too voluminous to analyze. First you filter the data down to only the steady state response interval. Usually by looking at the user load and the hit rate charts you can discover when a steady state was reached. You can double check this against the page response time versus time graphs to make sure the response times of interest have become stable.

By decomposing the transaction times into pages and then page elements that are taking the most time to process, you can see which operations are those most critical to tune. Taking a particular page element, you can then break down by method call the request that you want to analyze from an application tuning perspective. Depending on the frequency of calls as well as the time spent in each call, you might want to forward this data to the development organization for algorithm tuning.

A third type of experiment is used to verify the stability of the system intended for continuous operation without down time. Many enterprise systems have to be available 24 hours a day / 7 days a week with little if any *regular system down time*. To accept a new software version for one of these systems, a 3-7 day test of continuous operation at full system load is typical to ensure that the system is ready to be put into production. For this test you set up the statistical results (at the page level only) and operating system monitoring data to be sampled once every 5 to 15 minutes.

Test log data is completely turned off for this test. Running this test might require that you write special logged messages into a data file for every few minutes to verify that transactions are properly executing because normal logging is turned off. As long as the number of counters being sampled in the statistics model is only a few hundred, this mode should permit a several day test run. If your test runs with very few errors, you can also run the test logging set to *errors only* mode and determine the sampled number of users that you need error data from to get an accurate picture of a failure mode when it occurs.

The final type of experiment we discuss here is that of taking *final* capacity measurements. In this test run, you will have resource data being taken from each of your test agent systems to verify that you are in a *safe* operating region. In Performance Tester terms, this means that the CPU utilization on the test agents average no more than 70%, and there are no peak periods where the utilization hits peaks of over 90%. Because the memory allocation is constrained by the Java heap specified on the test agent, memory statistics should not be an issue.

If, however, you have multiple playback engines sharing a physical memory that cannot easily contain 100% of the Java processes used in its working set, then you might have to monitor paging and swapping on that test agent. In general, this is not a recommended test agent configuration because one playback engine can usually consume all available CPU resources on a system. Network and file I/O data rates should be monitored to make sure that the test agent is not constrained by limited I/O bandwidth and is unable to accept the server responses at full server output speed.

If there is a concern about the negative impact of Java garbage collection (GC) on the test agent, you can turn on the verbose GC logging (using the `-verbosegc -verbosegclog:C:\temp\gclog.log`) and view the length of time spent doing garbage collection. There are tools available through the IBM Support Assistant to analyze these logs. In general, this should not be a problem, unless you are running very close to the edge in heap size and doing frequent, unproductive garbage collection cycles.

When you have validated that the test agents are operating in a *healthy* operating region, you should verify that the transactions are operating properly and the steady state operating region has been identified and the report data filtered down to only this data. Based on this region, you can export the performance report and any other reports that have interesting data so that it can be included in a formal final report of your findings. Candidates include the transaction report (if those have been defined in the tests), the verification point report, and the HTTP percentile report (assuming that test log at the primary action level has been kept). The performance report by itself basically contains all the data necessary to show what the expected page performance of the system is. Statistically speaking response times are best reported as a 85th, 90th, or 95th percentile while throughput of transactions are best reported as averages completion times or system rates per second or per hour depending on their execution time.

2.2.10 Analysis and tuning

Analysis of the test results and system tuning is done in between each of the experiments. The experiment is designed to collect the data needed for tuning the subsystem or balancing the load across a cluster of servers. By moving from tuning a single application server to driving a cluster of application servers, you also introduce issues with pooling of connection to back-end database servers. Normal programming defaults will have multiple connections per virtual user session to the back-end database. This proves to be a significant and often repeated tuning exercise to redesign the application to pool these connections across multiple user sessions. Sizing the pool of connections to meet the need of the expected load of simultaneous outstanding requests to the database must be done while driving one or more of the application servers at full load.

It is important when planning your performance testing project to allow time in the schedule for sizing the application server, testing the load balancing across the cluster, tuning the database connection pool and maybe even some database table indices or optimizing certain application segments making multiple database queries in sequence to satisfy a user request. Leave time for possible work on heap fragmentation or memory leak issues in the application server testing if the application has not been stress tested before or if this is a significant re-write of the application. When these preliminary activities have been scheduled, then you can add in three to five full load tests to get your final system capacity measurements. Also if a multiple day soak test for system availability is requested, add schedule time for at least two or three attempts at performing this test with at least one day in between to analyze the results and modify the application in preparation for the next test run.

2.2.11 Performance test final report

Putting together the final report of your performance test project is very important to the long term value of the performance test project. Recording each of the experiments, summarizing the test results, the analysis, the corrective actions taken, and the post-tuning test results is crucial to growing the experience and expertise of the performance testing capability of your organization. Maintaining a lab log that contains all of the system and software changes introduced into the system during the entire project and the sequence of experiments can be beneficial from a change control perspective and also in a complete understanding of the final state of the system that corresponds to the final *run for the record* set of results.

The report should also detail out the complete system configuration including operating system tunables, disk volume layouts, size and characterization of the test database, network topology with firewall details, load balancers, and complete specifications of the hardware and software versions present on each of the systems. Application versions are critical to later understanding the use of these results as a baseline for future measurements or capacity estimates based on these test results. Performance Tester reports can all be exported to HTML, which can then be directly cut and pasted into a final report. This makes getting the raw results and operating system resource data into the final report a very efficient operation.

Probably the most important part of the final report is the enumeration of the testing goals with the associated success criteria as compared with the final test results. Stating clearly whether the system has passed the test goals should be explicit and summarized. Putting these project findings in an executive summary at the beginning of the report is recommended with the justification coming from the main body of the report. The final two sections describe the conclusions reached by the project and recommendations for future work.

As with any significant project, you should hold a post-mortem discussion with the project team members and gather both the good practices to continue and the troubling one that have to be adjusted for a better experience next time. This should not be part of the final report because it is an organizational and process asset rather than pertaining to the project content.

2.3 Performance testing artifacts

This performance testing methodology is built around the following artifacts:

- ▶ Workload specification
- ▶ Performance test plan
- ▶ Performance test tool assets
- ▶ Performance test results
- ▶ Performance analysis document

Each has its place in the projects movement from testing requirements to the ultimate analysis document containing summarized performance characterization and perhaps even recommendations for follow-on projects.

The workload specification along with the test plan constitute the statement of work for the performance test. The workload specification contains the content and scope of the test while the test plan discusses resources, schedule, and tasks required to obtain results defined in the workload specification. The test tool assets, test results, and performance analysis document can be considered the valuable output artifacts from the performance test project.

2.3.1 Workload specification

The workload specification is the most important document to help define the project and provide scope for the performance test. You can think of the workload specification as the requirements and design document for the test. It contains all of the important decisions that determine the workload model that is used to predict the intended production workload of the system. By constructing a workload model and gaining agreement among the stakeholders that it is accurate, you have what you need to implement the test and obtain the requested performance results.

A complete workload specification contains the performance goals of the test, the user scenarios, the measurements to be taken, and the playback dynamics, including user think times and input data variations. In addition, the maximum (or minimum) acceptable value for each response time (or transaction rate) measurement should be included so that there is a clear notion of when the system has adequate performance.

2.3.2 Performance test plan

The performance test plan describes the test laboratory setup as well as the test experiments that are required to answer all of the performance goals set forth in the workload specification. Each of the experiments list the testing steps planned and the output from the test.

The test plan also contains a test interval for each experiment and the order in which they should be performed. Together with the workload specification, the performance testing project can be sized and scheduled.

2.3.3 Performance test tool assets

The test tool assets used in completing a performance test project obviously vary according to which tool you are using. For the Rational Performance Tester product, the assets consist of a performance test project containing recordings of the test scenarios, tests generated and modified as needed for each scenario, workload schedules for each experiment, and datapools containing the input data so each virtual user can submit different data values.

2.3.4 Performance test results

At the completion of each test experiment, the tool generates a test result set. From these result sets, many different analyses can be performed resulting in standard or customized reports viewable within the Rational Performance Tester tool. When the analysis is complete, the reports should be saved for later viewing or for exporting so the data can be used in the performance analysis document. Exported report data from test result sets can also be considered independent test result assets.

2.3.5 Performance analysis document

The performance analysis document takes the test result sets and describes the important content from the results of each experiment. The system tuning recommendations made during the test plan phases as well as the final results from each test phase should be written into the document. The overall test goals as stated in the workload specification should be used to guide the organization of the results and the analyses described.

Probably the most important section of the performance analysis document is the conclusions from the overall performance test project and the suggested next steps drawn from the testing project. Normally, when the performance testing project is done at the end of a system development and prior to system deployment, the decision is whether or not the system is ready to be deployed. The test results and their analysis should be organized and emphasized to support the conclusions stated in the document.



Recording and test generation

This chapter describes how to configure the Web browser for recording of tests, how to record from a secure Web site, and how to get around problems when recording a test.

3.1 Configuring the browser before you record

Before you record, you have to:

- ▶ Set the application (browser to record).
- ▶ Verify your browser settings.
- ▶ Set Mozilla environment variables, if you are using Linux.

All modifications must be performed before Internet Explorer® (IE) is loaded by the recorder. Do not modify the Internet Explorer configuration settings during the recording. If the settings are modified during the recording, the recording will fail. When the recorder loads, it examines IE configuration settings and then tells IE to send the HTTP traffic to the port it is listening on (localhost:1080 by default).

3.1.1 Set the application to record

If you record with Internet Explorer, you do not have to set an application to record, because Internet Explorer is the default application.

If you record with Mozilla or Firefox, set the application to record as follows:

- ▶ Start Performance Tester.
- ▶ Select *Window → Preferences*, and expand *Test*.
- ▶ Select *HTTP Recording*, and set Application to Record to *Mozilla* or *Firefox*.
- ▶ Type the path to, or browse to, the browser executable.

If you record with another browser, set the application to record as follows:

- ▶ Start Performance Tester.
- ▶ Select *Window → Preferences*, and expand *Test*.
- ▶ Select *HTTP Recording*, and set Application to Record to *None*.
- ▶ Manually configure and load your browser. Consult the browser proxy configuration documentation on how to do this.

3.1.2 Verify browser settings

Make sure that your Internet options are set correctly before you start to record. To set Internet options on Internet Explorer:

- ▶ Select the Internet Explorer icon and *Properties*.
- ▶ In the Connections tab, click *LAN Settings*.

- If you are recording directly with Internet Explorer (that is, you are not using a proxy), clear all fields in the Local Area Network (LAN) Settings window (Figure 3-1).

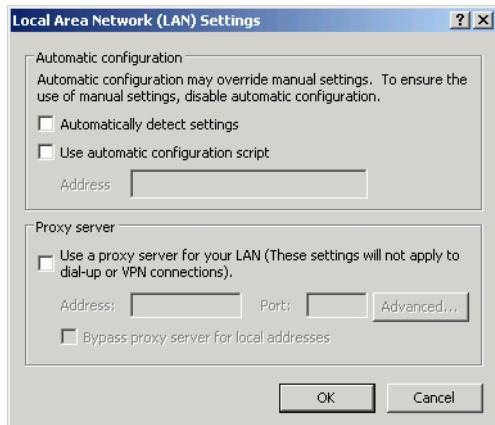


Figure 3-1 LAN Settings

- If you are using a proxy server with Internet Explorer:
 - Set the LAN Settings window the same way as in the previous step, except select *Use a proxy server for your LAN*. Be sure to clear *Bypass proxy server for local addresses* (Figure 3-2).

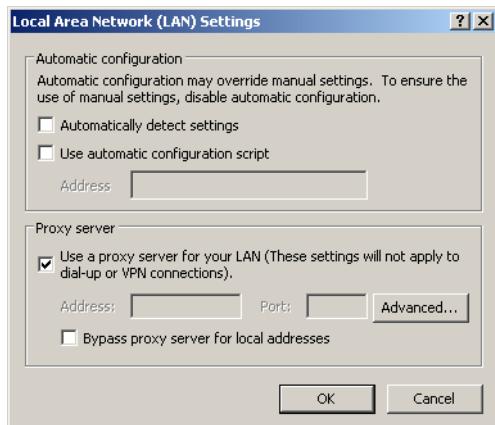


Figure 3-2 LAN Settings with proxy server

- Click *Advanced* and in the Proxy Settings window, and set the fields as shown in Table 3-1.

Table 3-1 Proxy settings

| Field | Value |
|--|---|
| HTTP | Points to the proxy server and port (typically the same address and port as the secure proxy) |
| Secure | Points to the secure proxy server |
| Do not use proxy server for addresses beginning with | Must be cleared |

- The Proxy Settings are shown in Figure 3-3, with information specific to your proxies.

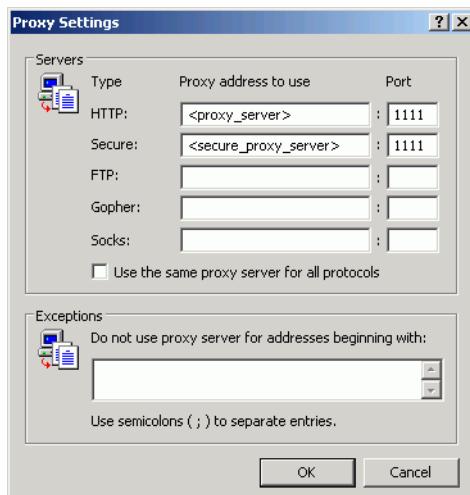


Figure 3-3 Proxy Settings

3.1.3 Set Mozilla environment variables

If you record with Mozilla on Linux only, set the Linux environment variable MOZILLA_FIVE_HOME to point to the directory that contains Mozilla. By default, Mozilla is installed in /usr/local/mozilla on most Linux computers. Therefore, to set the environment variable correctly, issue the following command before you start Performance Tester:

```
export MOZILLA_FIVE_HOME=/usr/local/mozilla
```

3.1.4 Verify that the Agent Controller is running

Verify that the Agent Controller is running on your local computer and on the remote locations that will add load to the schedule. To verify this:

- ▶ Select *Window → Preferences*, expand *Profiling and Logging*, and click *Hosts*.
- ▶ Your local computer is listed. To verify other computers, click *Add*, and type the host name (or IP address), the port, and then click *OK*.
- ▶ Select the computer to test, and click *Test Connection*. A message is displayed that says that Agent Controller is running or tells you why the connection failed.

3.2 Recording from a secure Web site

Rational Performance Tester supports the following authentication schemes:

- ▶ Basic
- ▶ NTLM

3.2.1 Disabling security warnings when recording

Performance Tester uses a proxy recording program that intercepts all traffic between the browser and the Web server. During recording at a secure Web site (with a Web address that starts with `https://`), by default you see a security warning before every action and must confirm your acceptance of a security risk to continue.

If you do nothing, this warning is displayed with every browser action, and you must repeatedly click *Yes* to continue. Performing the following procedure installs the recorder certificate on the local host as a trusted authority and thereby disables warnings from Internet Explorer during recording at secure Web sites. This procedure does not affect other browsers that record from secure Web sites—they will display warnings before every action.

To disable security warnings when using Internet Explorer to record from a secure Web site:

- ▶ During recording, the first time the warning is displayed, click *View Certificate*.
- ▶ In the Certificate window, click *Install Certificate*.
- ▶ In the Certificate Import Wizard window, click *Next*.
- ▶ On the second page of the wizard, click *Next*.

- ▶ On the last page of the wizard, click *Finish*.
- ▶ In the Root Certificate Store confirmation window, click *Yes*.
- ▶ In the window that reports that the import was successful, click *OK*.
- ▶ In the Certificate window, click *OK*.
- ▶ In the Security Alert window, click *OK*.

3.3 Non-browser recordings

Some Web sites start applets which generate HTTP traffic that the customer wants recorded. When writing an applet that generates HTTP traffic, the customer must either follow IE's Internet settings or use the socks protocol and send the traffic to port 1080 on the localhost. HTTP traffic will not be recorded if is not sent through the port we are listening on.

3.4 Reconfiguring IE after recording is interrupted

Browser options are changed during recording and are reset after recording is complete. If you interrupt recording, the browser stays in its changed state, and you might receive The page cannot be displayed messages. To fix this, reset the browser to its initial state.

To return your browser to its initial state:

- ▶ Select the Internet Explorer icon and *Properties*.
- ▶ In the Connections tab, click *LAN Settings*.
 - If you do not use a proxy, in the Local Area Network (LAN) Settings window, clear *Use a proxy server for your LAN*.
 - If you use a proxy:
In the Local Area Network (LAN) Settings window, select *Use a proxy server for your LAN*, and then click *Advanced*.
- ▶ In the Proxy Settings window:
 - Add the proxy address and port number to the HTTP and Secure fields.
 - Remove the proxy address and port number from the Socks field.
 - In the Proxy Settings window click *OK*.
- ▶ In the Local Area Network (LAN) Settings window, click *OK*.
- ▶ In the Internet Properties window, click *OK*.

3.5 Troubleshooting

Problems in recording usually involve the following areas:

- ▶ The browser is configured incorrectly:
 - Remember to clear *Bypass proxy server for local addresses*.
 - *Automatic configuration* must not be enabled.
 - The Proxy Settings Exceptions list must be blank.
- ▶ Changes have been made to the browser during recording.
- ▶ The Agent Controller is not running, or there are problems with it.

3.5.1 Agent Controller problems

In this section we discuss some typical Agent Controller problems.

servicelog.log

To perform a HTTP recording, the Agent Controller must be running. If the `servicelog.log` cannot be found in the correct location (for example: `C:\Program Files\IBM\SDP70Shared\AgentController\config`), then the Agent Controller is not running and you will not be able to record. You will have to start the Agent Controller before recording.

If the `servicelog.log` is in the correct location, but you still cannot record, then you should review the entries in this file. Sometimes you can find the source of the problem by reviewing this log file.

Here is an example of a log file entry caused by having a reference link to a non-existent file in the `serviceconfig.xml` file:

```
Recording started
Connecting to agent controller on localhost/port 10002
IWAT3031E Start recording aborted due to exception:
org.eclipse.hyades.execution.core.DaemonConnectException.
```

serviceconfig.xml

This file contains the parameters used when starting the Agent Controller. The file should be set up correctly during the install process. However, sometimes you will have to review this file or modify it to get the Agent Controller to work successfully. You can modify this file by using an editor or by executing `setconfig.bat` (or `SetConfig.sh` on Linux) in the `~\AgentController\bin` directory. With this command-line utility, you can reset the path to the Java runtime, network access, and encryption settings for network communication to the Agent Controller.

3.5.2 Proxy recorder architecture

The architecture and connections for recording with a proxy server are shown in Figure 3-4.

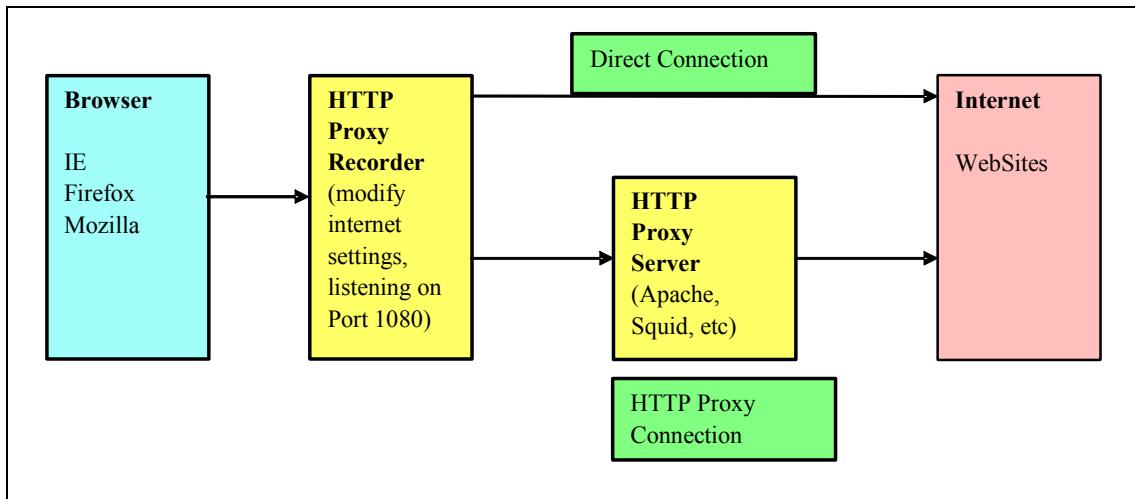


Figure 3-4 Architecture when recording using a proxy

3.5.3 Interpreting a .REC file (recording file)

Many power users of performance testing tools like to understand the file format of the recording file so they can examine the raw network communication messages that were passed to and from the server. The Rational Performance Tester stores the recorded data in an XML file.

There are a few things that you might want to extract from the first two sections of the recording (*.rec) file. If a problem occurs, the development team helping to solve the problem will ask for the recorder version (see the <recorderversion> line) and whether the recording is going through an HTTP proxy (see the <ProxyEnabled> line). The rest of the information will not be useful to most people.

Example 3-1 shows a typical example of the first two sections of a .REC file.

Example 3-1 Sample recording file

```
<?xml version="1.0" encoding="UTF-8"?>

<TRACE>

<TRCRecorderInfo type="start" ticket="1" timestamp="0">
    <recorderport>1080</recorderport>
    <date>11/15/2006 15:22:07</date>
    <recorderversion>4.2.0.8</recorderversion>
    <javavmversion>2.3</javavmversion>
    <javavmvendor>IBM Corporation</javavmvendor>
    <javahome>D:\Program Files\IBM\SDP70\jdk\jre</javahome>
    <osname>Windows XP</osname>
    <osversion>5.1 build 2600 Service Pack 2</osversion>
</TRCRecorderInfo>

<TRCRecorderInfo type="WEBBROWSERCONFIG" ticket="2" timestamp="0">
    <date>11/15/2006 15:22:08</date>
    <ProxyEnabled>FALSE</ProxyEnabled>
    <ProxyAddress>NONE</ProxyAddress>
    <ProxyPort>NONE</ProxyPort>
    <ProxyOverride>NONE</ProxyOverride>
    <ProxyAutoConfigURL>NONE</ProxyAutoConfigURL>
</TRCRecorderInfo>
```

3.6 HTTP recording tips

When the recorder is engaged and is recording browser or HTTP application interactions with a server, there are several recording tips that can help you get a robust recording and resulting generated test. Keep in mind that the network recorder is time-stamping each HTTP network packet sent to the server or received from the server. The order, interleaving, and delays between these packets is significant to the test generation algorithms. The content of the browser file and cookie caches have an impact on the traffic as well.

3.6.1 Browser caching

The browser that you use for recording will be mimicked in the test as far as number of parallel socket connections to the server and which connection is used for each request. Therefore it is important to record using the browser and cache state that best represents the expected behavior of the user community you are modeling in the workload.

We recommend that you always flush the cookie cache prior to recording to add the proper construction of the cookie cache during playback. This might not be true for the file cache, however.

For example, if the Web site that you are visiting is frequently visited by the user community and most images would be present in the user's file cache, then you should run through the user scenario prior to doing the actual recording to pre-populate the browser file cache. Then when you record your scenario, the file cache already contains most of the images and responses accordingly during the recording. This behavior will cause a different traffic pattern to the server, including the browser performing a conditional GET to the server to see if the images have been updated and receiving a 304 Not Modified response but not the actual image data.

3.6.2 Overlapped requests and responses

One of the worst problems that you can experience is due to rushing through your recording. Because the interleaving of requests and responses is tracked and used by the test generator to identify user actions and page boundaries, you must refrain from clicking the next user action before the page is completely received from the server. Most browsers have a status indication and a visible animation to indicate that it is still waiting for response data from the server. Some of this content might actually be destined for fields not present in the scrolling window visible on your screen.

If you truly want to measure the entire server response including the page components that are not visible, then you should wait until the browser indicates the server has returned all the data and is done. By not observing this tip, you might get the page content of page one mixed into page two, and neither page response time accurately measures the real page response time. Because the whole concept of an HTTP page is an artificial boundary condition not specified in the protocol, you can help line up the user's concept of a page with the recorded user scenario steps by waiting so that successive page contents are not interleaved.

3.7 Test generation

Each of the supported protocols and test environments have their own recording and test generation capabilities. For the Citrix and SAP environments, refer to the chapters on those environments. For HTTP tests, there are a number of test generation options that can impact the way the test is generated. This includes whether to perform the optional test generation logic based on the decoding of Siebel® or SAP specific Web traffic.

If this logic is used and the run-time test content is generated using these options, the protocol extension license token will be checked out during multi-user playback for that environment. If the extension license token is not present (or available due to some other user), the test playback can not be performed.

For certain portal environments where the specialized Siebel or SAP content is not being tested, you might be able to turn off the specialized decoding at test generation time so that a protocol extension license is not required for test playback. However, if that content is being tested, you should leave the test generation on automatic so that the specialized decoding is done and the application will play back correctly.

3.7.1 HTTP test generation

In addition, Performance Tester uses the user's delay time between HTTP requests to help create these page boundaries. If you delay at least five seconds between the end of a page response before you click the next user action, then the test generation will create a separate page with a user think time. This five second delay is a user preference that can be modified. Many users find that setting this value to two seconds for pre-scripted or familiar user scenarios provides better page splitting.

Select *Window → Preferences* to open a dialog with a set of Eclipse preferences (Figure 3-5):

- ▶ Select *Test → Performance Test Generation* in the left tree control.
- ▶ Select the *Protocol* tab on the right side and you can set the user preferences associated with HTTP test generation.
- ▶ You can adjust the *Generate new page if delay between requests* option to change the page split delay value.
- ▶ You can also change the SAP Web or Siebel decoding options from *Automatic* to *Off* to turn off these enhanced capabilities.

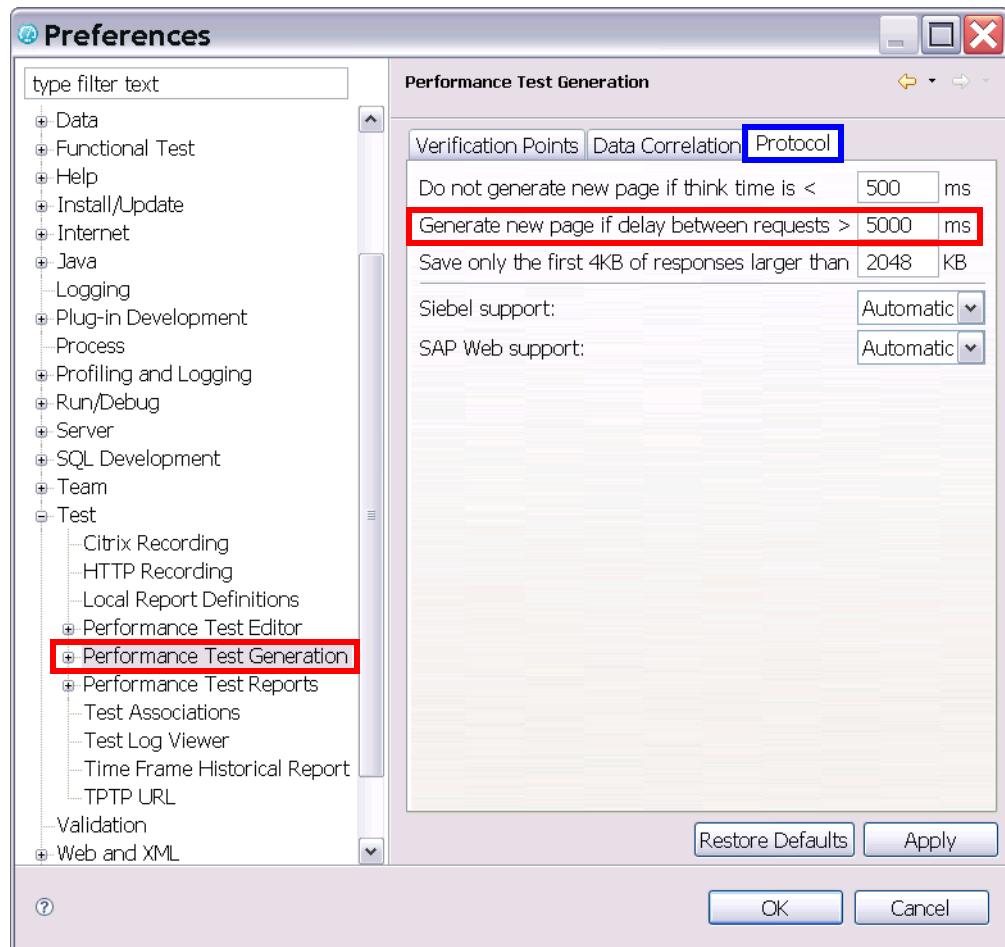


Figure 3-5 Performance test generation preferences for protocols

By making use of these recording tips and tailoring these test generation settings to meet the needs of your recording environment, you will get automatically generated tests that are closer to the result you want, and will require less editing afterwards.



Test editing and augmentation

This chapter describes the various ways of editing and augmenting tests in IBM Rational Performance Tester. Here we discuss the different reasons for modifying tests as well as the basic steps for doing so.

To properly edit a Performance Tester test, you must first thoroughly comprehend the structure and workings of a test, as well as the purpose of the particular test you are editing. We recommend that you have a good understanding of the topics covered in Chapters 1 through 7 of this book before you begin editing tests.

We cover the following topics:

- ▶ Editing to complete or correct a test for playback
- ▶ Enhancing a test to extend its initial capability
- ▶ Modifying a test for changing testing needs
- ▶ Differences between editing different types of tests

4.1 Overview of test editing

Editing tests in Performance Tester is done entirely through the graphical test editor, with the exception of editing custom code (see 4.7.6, “Custom code” on page 116). Some test editing and augmentation is almost inevitable, however the extent of what and how much is changed in a test can vary widely.

There are a number of reasons why you might want to edit and augment a test in Performance Tester. You would edit a test to:

- ▶ Associate test data with a test
- ▶ Manipulate data correlation
- ▶ Verify system functionality during test execution
- ▶ Modify or control test flow
- ▶ Fulfill measurement and reporting needs
- ▶ Modify or manipulate timing or connections
- ▶ Improve test readability and reuse

The various sections of this chapter explain how to do these tasks for the different types of tests in Performance Tester.

4.1.1 What you need to know about editing tests

A single test in Performance Tester is composed of several components and files, most of which you do not even have to know about. While the tests are ultimately coded in Java, you will not directly modify the Java code. In general, you should **not** edit any of the test files through a view such as the Eclipse Navigator view or directly through the file system. Editing and augmenting tests is always done through the default *test editor* in Performance Tester.

4.1.2 Editing and augmenting tests versus schedules

You want to edit tests to achieve some specific performance testing purpose. Manipulating test schedules, which are discussed in Chapter 6, “Scheduling tests” on page 155, is another way of achieving specific testing purposes. In some cases, you could edit either the test or a test schedule containing the test to achieve the same objective. For example, you could repeat a certain transaction by adding a loop to either a test or to a schedule. To determine when to modify a test versus modifying a test schedule, you must understand and consider the purpose of each, the relationship between them, and how you are utilizing these test artifacts within your project.

In general, you would manipulate a test to create an item that reflects the smallest test execution asset. The tests then become the building blocks used to create the schedules for test execution. Simulating workload analysis is generally done more inside schedules than in tests. It is easier and more common to develop multiple variations of a schedule for multiple test runs during performance testing.

Before you begin editing a test, you should consider how you could reuse the test for testing goals. You might want to create another test based on the first, or you might want to implement your preferred test functionality in a schedule, if that is possible. Because you typically reuse tests within many schedules, it might not make sense to continue to significantly modify tests well into the test execution phase of a testing effort. Making major changes to a test increases the odds that you would also have to edit every schedule that contains the test.

4.1.3 General rules when editing

There are many guidelines and user interface conventions that apply to all editing done in Performance Tester.

Double-click to open test artifacts for editing

To open a test or other test artifact for editing, simply double-click it from the Test Navigator (the default location is on the left side). This will open the test in the test editor; other artifacts such as datapools and schedules will also open for editing. The test editor is composed of the Test Contents area on the left and Test Element Details on the right. Selecting different elements from the Test Contents will display different detail information in the Test Element Details area.

Figure 4-1 shows the general process for editing parts of a test.

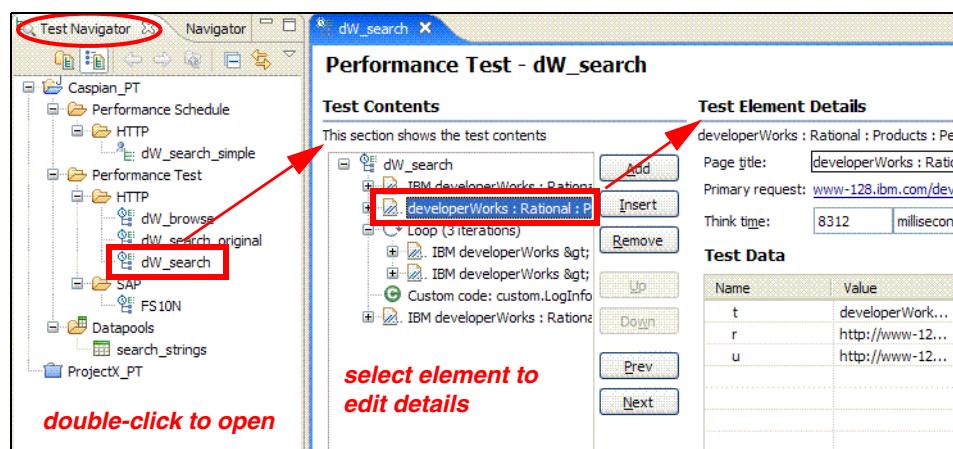


Figure 4-1 Test navigator and test editor

Italics indicates changes not saved

Whenever you change a test element, the element text and the test name text change to *italics*; an asterisk will also appear next to the name on the editor tab. The text will remain in italics until the changes are saved (Figure 4-2).

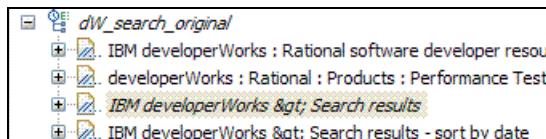


Figure 4-2 A test with unsaved changes

Limited undo capability

In general, the changes you will make to a test artifact in Performance Tester do not have an *undo* capability. However, most changes are not saved until you deliberately save them, so you can reverse unwanted changes by closing an artifact without saving. Using the italics indicator, you can stay aware of what has changed and when changes are saved to control any unwanted edits.

Limited copy/paste

Because artifacts and elements in Performance Tester are more than simple text or Windows objects, you cannot use copy and paste functions to replicate them. You can, however, replicate tests using the *File → Save As* feature. The menus in Performance Tester indicate whether copy/paste is possible or not.

4.1.4 Colors and test editor preferences

Performance Tester uses colors in the test editor to convey information. You should become familiar with these as this can help you edit and develop tests faster. You can also change the default colors if you want (Figure 4-3).

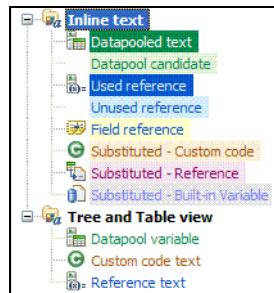


Figure 4-3 Default colors used in test editing

There are several preferences that affect default behavior when editing tests. You can modify these settings to better suit your needs. Note that these are user preferences, and changes will only apply to a particular user/workstation.

To change the test editor preferences and colors:

- ▶ In the Performance Tester menu, select *Window → Preferences*.
- ▶ In the Preferences Tree browser on the left, navigate to *Test → Performance Test Editor* (Figure 4-4):
 - The General tab contains settings for control behavior.
 - The Colors and Fonts tab allows you to change colors as previously mentioned.
 - You can expand the Performance Test Editor node in the tree browser to display other preference for specific types of tests, such as HTTP or SAP.

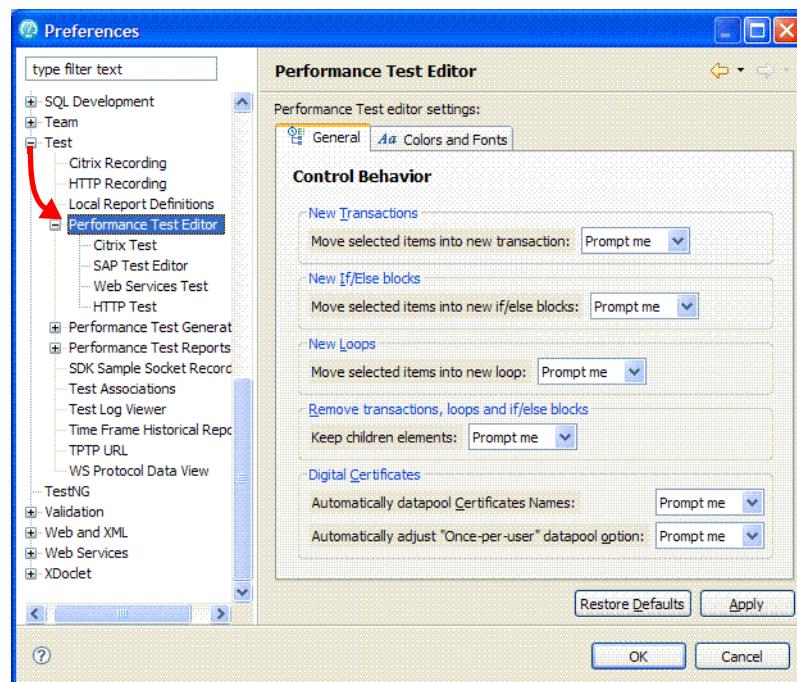


Figure 4-4 Performance test editor preferences

For more information on recommended practices for test editing, see the section that follows.

4.2 Test editing best practices

Performance Tester provides great flexibility in editing tests and test artifacts, but there are several best practices that can improve the efficiency and effectiveness of using Performance Tester.

4.2.1 Projects and folders

Organization of test assets is crucial to a good testing effort, so where you save your tests, datapools, and other test artifacts in Performance Tester is very important. While the logical test navigator view will group and separate tests, schedules, and datapools, you might also want to create additional folders to categorize and group artifacts based on the needs of the specific testing effort.

Every time you create a new test artifact in Performance Tester, you have to select a folder location to store it. Because you can have more than one project open in Performance Tester, you might also have to select the correct project and folder.

Likewise, when you are associating an artifact with another, such as selecting a datapool for a test, or selecting a test to put into a schedule, you have to browse different folders and possibly different projects to select the correct asset. You must be careful to select the correct project and folder, especially for larger and more complex testing efforts.

You can rename test assets and move them to other folders or projects if needed. For medium to large testing efforts, you should document the naming conventions and test asset organization strategy.

4.2.2 Test search and replace

One of the most important available for editing tests is the test search feature. This is important for data substitution as well as other editing described in this chapter. This function in Performance Tester offers more capability and is more complex than common search-replace functions found in many applications such as word processors.

A test search will look for text in any part of the test such as requests, responses, or other element fields specific to the type of test. There are many options that can filter the search. The test results are shown in a view similar to the Test Contents editor and you can jump to the location of each match from the results. Search results also persist until the next search, so you do not have to keep searching for the same item over and over again while editing the test.

Searching

To search for items within a test:

- ▶ Open the test.
- ▶ In the menu bar select *Search → Search*.

Note: If you use the *Search*  toolbar icon, then you must ensure that the *Test Search* tab is selected (Figure 4-5).

- ▶ In the *Search for text* field, type the text that you want to locate.

You can leave this field blank, depending on your search strategy. For example, if you know that a string occurs in elements or element instances that you are not interested in, using the options described below, you can locate the elements or element instances of interest before entering the search text into this field.

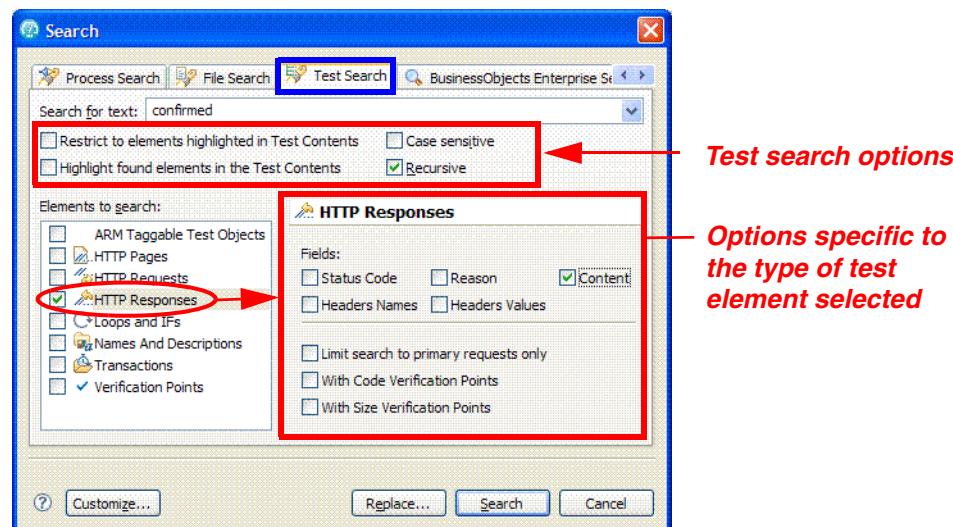


Figure 4-5 Test search: Searching HTTP response content

- ▶ Select the test search options:
 - To search only in elements that are selected in the Test Contents area, select *Restrict search to elements highlighted in Test Contents*.
 - To do a case-sensitive search, select *Case sensitive*.
 - To have found elements highlighted in the Test Contents area, select *Highlight found elements in Test Contents*. You can use this option with *Restrict search to elements highlighted in Test Contents* to designate the element instances of interest before specifying the text of interest.

- To have the search through all child element levels of the test from the test element selected, select *Recursive*.

Selecting *Restrict search to elements highlighted in Test Contents* restricts the types of elements that you search in this step to those that are selected in the Test Contents area. For example, if you select *HTTP Pages* here and only one page is selected in the Test Contents area, only matches in that page are found. Otherwise, if the test name is highlighted and *Recursive* is selected, then string matches in every test page are found.

- ▶ Select the options specific to the type of test element selected. Under *Elements to search* select each type of element you want to search.

When you select an element under *Elements to search*, the lower-right area of the Search window will change to display the options for that type of element. These options are different for each type of element and they allow you to define how an element should be searched. Figure 4-5 on page 47 shows the options for an *HTTP Responses* element, whereas Figure 4-6 here shows the options for an *SAP Element*.

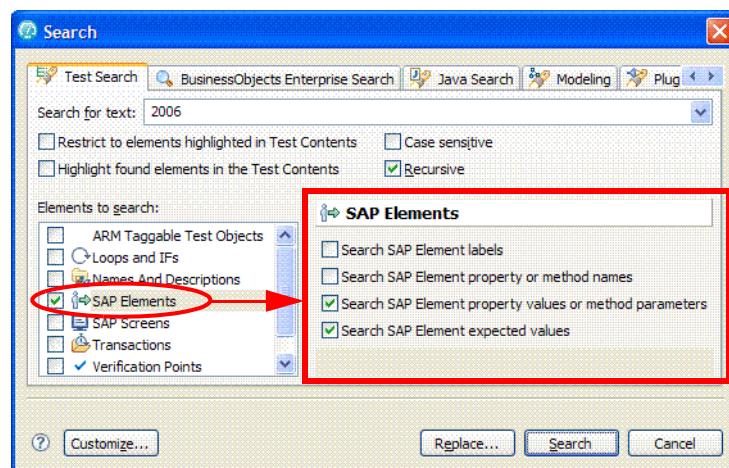


Figure 4-6 Test search: Searching SAP elements

- ▶ Click *Search*. The results of the search are displayed in two views: the *Search* view, which lists the objects that contain matches (Figure 4-77), and the *Test Search Match Preview* view (Figure 4-9 on page 50), which displays the matches that were found.

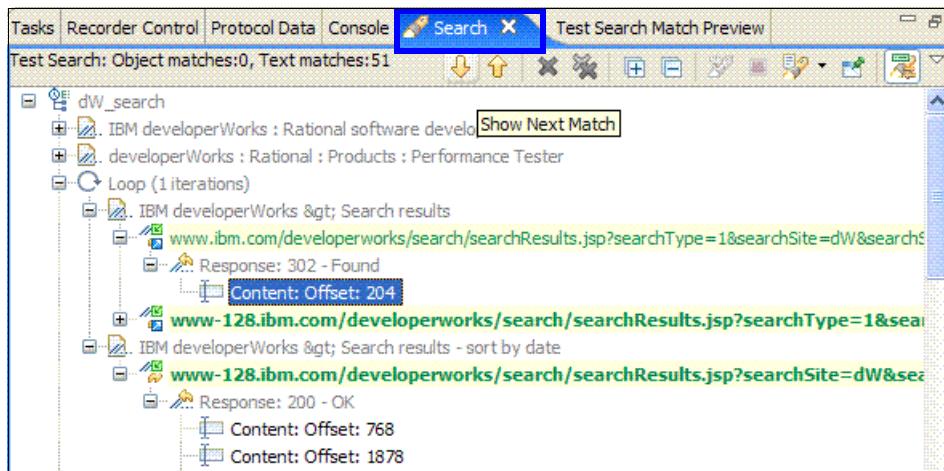


Figure 4-7 *Search results: Search view*

Search results

When you have results from your search, there are several things you can do. Most of these are available by right-clicking in the Search view (Figure 4-8).

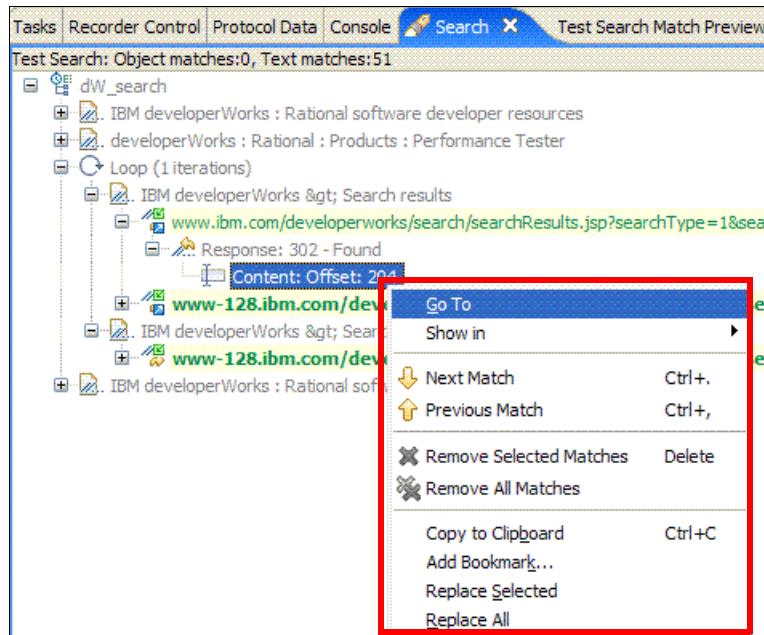


Figure 4-8 *Search results view: Search actions*

- To preview a found string in the *Test Search Match Preview* (Figure 4-9), click the object.

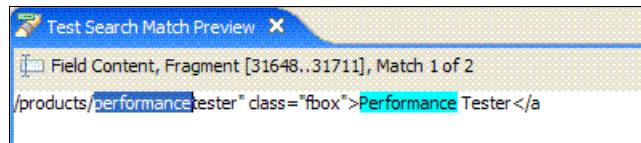


Figure 4-9 Search results: Test search match preview

- To open your test at the location where an instance was found, double-click the object. Alternately, you can right-click and select *Go To*.

This will switch focus to the test editor, expand the Test Contents to the match location, and highlight the found search item. From here you can make whatever edit is needed in the test.

Tip: With HTTP response content, after using the *Go To* function, you can then select the *Protocol Data - Browser* tab to view the rendered HTTP page. This can be done to further validate that you have located the intended value.

- To perform a different search action, such as proceed to the next or previous match, bookmark a test location, or replace values, right-click the object and select your choice.

You can perform a number of searches while enhancing a test, and it is often useful to return to previous search results. Performance Tester will keep a history of test searches within an Eclipse session (if you exit the application and restart the history is cleared). You can access the history by clicking *Show Previous Searches* (Figure 4-10).

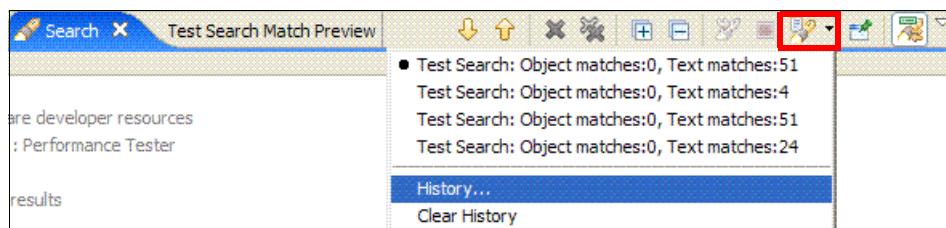


Figure 4-10 Show previous searches: Search history

Replacing values

There are several ways to replace values in a Performance Tester test. This is useful for modifying the environment or other configurations that you will execute your test will be executed tests in (see 4.8.3, “Changing environments for testing” on page 128).

To replace items within a test:

- ▶ Open the test.
- ▶ In the menu bar, select *Search → Search*.
- ▶ From the Test Search window, set your search options as described in “Searching” on page 47.
- ▶ At the bottom of the Search window (not the search results view), click *Replace*. This will execute the search and open the *Replace* window (Figure 4-11).

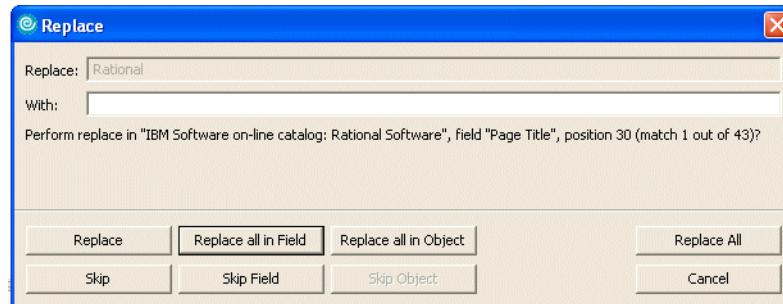


Figure 4-11 Search and replace

- ▶ Enter the replacement value in the *With* field.
- ▶ Click the appropriate button to select the replacement action. If you are selectively replacing text, click *Replace* or *Skip* until all found instances have been displayed.
- ▶ Alternately, you can replace values from the search results view by selecting a match, then right-clicking and selecting *Replace Selected* or *Replace All* (Figure 4-8 on page 49). This will open the *Replace* window and you can replace items as described in the previous step.

4.2.3 Maintaining variations of a test

Because you are likely to edit a given test repeatedly throughout a testing effort, it is important to have the ability to return to a previous version of a test.

Using a configuration management tool such as IBM Rational ClearCase® LT provides the most powerful functionality for this. However, in a testing effort with short time constraints, this might not be the most time effective way to deal with the need. A simpler and faster way to do this is by copying and saving different variations of the same tests. This approach does not provide true version control and is subject to user error.

An ideal strategy is to use both methods as needed; saving copies for minor and frequent variations and a configuration management tool for periodic version control.

Saving copies of tests

The test that you edit in Performance Tester is usually generated from a test recording. You can therefore use the recording as the original version and save the test with different names for different modified versions. The most common naming strategy for this would be to add a suffix at the end of the names for each version of the test. For example, you might have two tests named “lookup_initial” and “lookup_enhanced”.

To save a copy of a test:

- ▶ Open the test you want to copy.
- ▶ Select *File* → *Save As*.
- ▶ In the *Save As* window, select the project folder you want to save the new test into. Enter a *File name* different than the original and click *OK*.

Using ClearCase

IBM Rational ClearCase LT can provide version control of test assets through the Rational ClearCase SCM Adapter in Eclipse. This provides an easy interface built into the same Eclipse shell as Performance Tester for checking in and checking out performance test assets such as tests, datapools, custom code and others. This capability requires the purchase of Rational ClearCase LT licenses and the installation of client software on the Performance Tester workstation.

More information on Rational ClearCase LT can be found at:

<http://www.ibm.com/software/awdtools/clearcase/cclt/index.html>

4.3 Editing different types of tests

Performance Tester can generate several different types of tests based on the network protocol being recorded and emulated. Editing of tests is similar for all types of tests, but there are some differences and additional functionality provided for some types. The following sections explain the key differences between editing each of the primary types of Performance Tester tests provided by IBM Rational.

Note that the IBM Rational Performance Tester Extensibility Software Development Kit (SDK) enables you to create extensions to support new protocols without modifying the core product code. The SDK has been used to develop all of the non-HTTP protocol environments, including support for the Oracle® eBusiness Suite (created by an IBM business partner), the Citrix extension, the SAP extension, the SOA Quality extension, and the SDK sample socket recorder.

Running some test types requires additional licensing beyond the base Performance Tester license, which includes HTTP/HTTPS. In the following sections, 4.3.2, “Editing Siebel tests” on page 55 through 4.3.4, “Editing Citrix tests” on page 62, only apply to installations which have the extensions to enable these protocols.

4.3.1 Editing HTTP tests

Web-based applications communicating with HTTP and HTTPS are the most common type of systems that are performance tested today. For this reason, this is the type used in almost all tutorials, training exercises, and examples. HTTPS is simply HTTP using Secure Sockets Layer (SSL) to encrypt and decrypt data for secure transactions. For more on handling SSL digital certificates with Performance Tester, see “HTTP Digital certificate overview” on page 131.

Figure 4-12 shows a sample HTTP test.

Test Contents

This section shows the test contents

- dW_search
 - IBM developerWorks : Rational software
 - developerWorks : Rational : Products : P
 - Loop (3 iterations)
 - IBM developerWorks > Search res
 - www.ibm.com/developerworks/s
 - Connection
 - Response: 302 - Found
 - www-128.ibm.com/develop
 - Response: 200 - OK
 - www.ibm.com/common/v14/print
 - www-128.ibm.com/developerwo
 - www.ibm.com/v14/buttons/go.
 - www-128.ibm.com/developerwo
 - www-128.ibm.com/developerwo
 - stats.www.ibm.com/rc/images/u
 - www.ibm.com/v14/rules/blue_r
 - IBM developerWorks > Search res
 - Custom code: custom.LogInfo
 - IBM developerWorks : Rational software

Test Element Details

www.ibm.com/developerworks/search/searchRes
WsearchScope=dWquery=datapoolsSearch.x=1

Request Attributes

| | | | |
|----------|--|---------|-----|
| Version: | 1.1 | Method: | GET |
| Host: | www.ibm.com | | |
| URL: | /developerworks/search/searchResults hSite=dW&searchScope=dW&query= | | |
| Data: | [Empty] | | |

Request Headers

| Header Name | Value |
|-----------------|-----------------------------|
| Host | www.ibm.com |
| User-Agent | Mozilla/5.0 (Windows; U; W |
| Accept | text/xml,application/xml,ap |
| Accept-Langu... | en-us,en;q=0.5 |
| Accept-Encoding | gzip,deflate |
| Accept-Charset | ISO-8859-1,utf-8;q=0.7,*; |
| Keep-Alive | 300 |
| Connection | keep-alive |
| Referer | http://www-128.ibm.com/de |
| Cookie | ibmSurvey=1166019214730 |

- Click to set as primary Delay:

Character set:

Figure 4-12 Sample HTTP test

The contents of an HTTP test is broken down into one or more pages, then individual requests with responses. A page will almost always correspond to a visible page rendered in a browser. Within the test editor, you can:

- ▶ Select values for datapools in the Test Data or Test Element Details (refer to 4.4, “Providing tests with test data” on page 64 for detailed steps on how to do this).
- ▶ Select or modify correlated values in the Test Data or Test Element Details (refer to 4.5, “Manipulating data correlation” on page 81 for detailed steps).
- ▶ Add verification to all or part of the test by right-clicking in the Test Contents (refer to 4.6, “Adding verification to tests” on page 89 for detailed steps).
- ▶ Insert test elements by right-clicking in the Test Contents (refer to 4.7, “Adding test elements” on page 105 for detailed steps).

- ▶ Modify connection parameters in the Test Element Details (refer to 4.8, “Adjusting connections and timing” on page 120 for detailed steps).
- ▶ Edit page names and add comments for readability in the Test Contents and Test Element Details (refer to 4.9, “Improving test reuse and documentation” on page 135 for detailed steps).

Except where noted, all editing and augmentation described in this chapter apply to HTTP tests.

4.3.2 Editing Siebel tests

Siebel applications run across the HTTP transport layer, therefore Siebel tests in Performance Tester are simply HTTP tests with several added features (Figure 4-13).

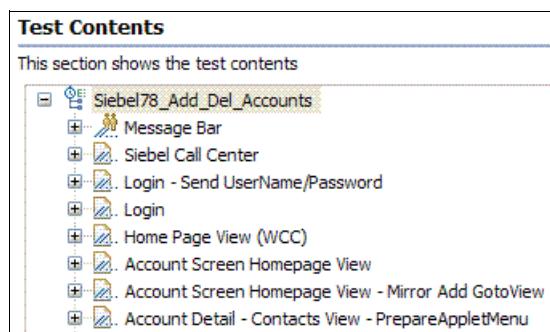


Figure 4-13 A Siebel HTTP test

For the most part, editing Siebel tests is no different than HTTP. The differences between Siebel and HTTP tests have primarily to do with test set up, recording, and generation, but some differences also apply to editing.

The first difference between editing Siebel tests and HTTP tests has to do with page naming and readability. Siebel tests are broken down into pages and requests just like those recorded from a Web-based application, but with the following differences:

- ▶ The first page of a Siebel test is named Message Bar, which emulates the ticker-tape message that Siebel application pages display. This “page” can be configured to repeat at any frequency you choose.
- ▶ Page names are generated to have more descriptive names and reflect the displayed screen. Therefore, they typically do not require renaming for readability (refer to 4.9, “Improving test reuse and documentation” on page 135).

Other differences between Siebel tests and HTTP tests have to do with configuring variable test data and correlation. Siebel tests will have improved identification of datapool candidates, provide data correlation for Siebel specific data structures, and include additional types of built-in Siebel data sources for correlation. In many cases, datapool substitutions are the only changes that you have to make to a Siebel test without needing to manually correlate any other values.

Identifying test data for datapools

The Siebel Data Correlation engine used during test generation will recognize Siebel-specific parameters which are generally of no interest to the tester and filter them out of the Test Data displays. The resulting datapool candidates in Siebel tests are more clearly identified, making configuring datapools for a test easier. Setting up a datapool for a Siebel test follows the same procedures described in “Providing tests with test data” on page 64.

Siebel star arrays and data correlation

Siebel variables are stored in a proprietary data structure called a star array. These are implemented in text over HTTP with the format: <length>*<data>. You can choose to substitute from a value in a star array the same way you would correlate any value (refer to 4.5, “Manipulating data correlation” on page 81). When you do this, you are asked whether you want to use Siebel or Generic correlation. Siebel correlation ensures that the <length> field is set to match the substituted <data> length during playback (Figure 4-14).

| Test Data | | |
|--------------|---------------|------------------------------------|
| Name | Value | Substituted with |
| SWEC | 10 | 10 SWE Counter (Current: 10, M...) |
| SWERowId | 1-1GBWB | "RowID" ("1-1GBWB") |
| SWETS | 1132224523500 | Timestamp - ("1132224523500") |
| SWEVI | | "" - Content |
| SWSECancelID | 1132224523 | Territory ("") |

Figure 4-14 Siebel data correlation

Siebel built-in data sources and correlation

The Siebel Web client produces several parameters that are passed to the server. These parameters are such things as timestamps and incremental counters. Performance Tester implements four built-in data sources that are used by the test generator to reproduce this behavior during playback (Figure 4-15).

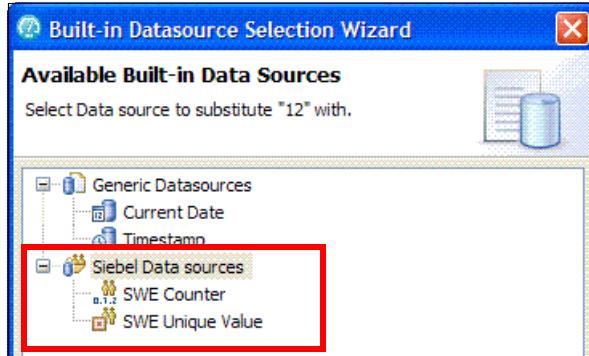


Figure 4-15 Siebel built-in data sources

These built-in data sources can be used independent of any Siebel tests as well by manually correlating any request data with the data source. The Siebel Data Correlation Library enables Performance Tester to automatically correlate Siebel-specific data. These correlations are designated with a unique icon in the test editor.

When you are editing a Siebel test, you can manually correlate request values with a built-in Siebel variables. To correlate a request value with a built-in variable:

- ▶ Open the test.
- ▶ Locate the value that should be replaced by a built-in variable.
- ▶ Highlight the value, then right-click and select *Substitute from → Built-in data sources*. The Built-in Datasource Selection wizard displays the types of variables that can be substituted.
- ▶ Select the type of variable and click either *Next* or *Finish*.
 - If you select Current Date, click *Next*, select the date format then *Finish*.
 - If you select SWE Counter, click *Next*, enter values for the counter in the Current Value and Maximum Value fields, and then click *Finish*.

Verifying Siebel specific content

A final difference between editing Siebel tests has to do with functional verification. Content verification (discussed more in “Siebel response content” on page 96) includes built-in Siebel error messages, in different languages, to select from. You can also add user-defined content to verify additional content.

4.3.3 Editing SAP tests

SAP applications use a proprietary protocol, commonly called SAPGUI protocol, so SAP tests in Performance Tester are very different from HTTP tests and editing them has a number of differences. The biggest difference is the SAP test editor, which breaks tests down into transactions and screens. The SAP Protocol Data view is also used to view SAP screen snapshots and GUI objects and to edit events. In addition to server requests and responses, there are also get events that retrieve data from objects in the SAP GUI. Like HTTP tests, you can still replace recorded test values with variable test data, add verification points, and test elements (Figure 4-16).

See Chapter 11, “Testing SAP enterprise applications” on page 383 for more information on testing SAP applications.

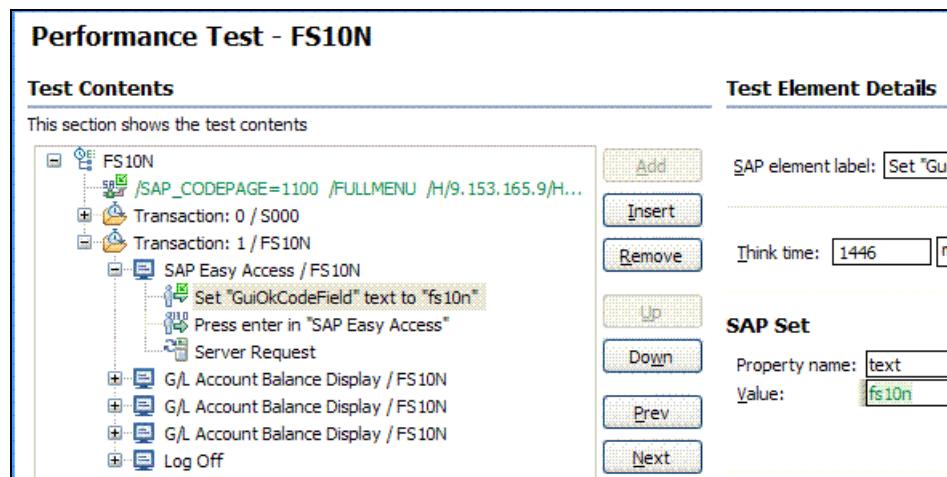


Figure 4-16 An SAP test

SAP test editor

For an SAP test, the test editor lists the SAP transactions by title, with screens, then GUI events within the hierarchy. Within this test editor, you can:

- ▶ Select values for datapools in the Test Element Details (refer to “Providing tests with test data” on page 64 for detailed steps).
- ▶ Select or modify correlated values in the Test Element Details (refer to 4.5, “Manipulating data correlation” on page 81 for detailed steps).
- ▶ Add verification to SAP events or GUI fields in the test:
 - With SAP tests you do this from the SAP Protocol Data view, which is different from the steps described in 4.6, “Adding verification to tests” on page 89.

- Refer to 4.6.2, “Verifying expected behavior in SAP tests” on page 96 for detailed steps.
- ▶ Insert test elements by right-clicking in the Test Contents:
 - Only transactions, screens, or SAP events can be contained in a loop or conditional test element.
 - Transactions can be added to contain SAP screens or other transactions.
 - Refer to 4.7, “Adding test elements” on page 105 for detailed steps.
- ▶ Modify some SAP connection and timing parameters:
 - Modifying connection and timing in SAP tests is different from the steps described in 4.8, “Adjusting connections and timing” on page 120.
 - Refer to “Editing SAP connections” on page 127 for detailed steps.

SAP Protocol Data view and GUI data

You can use either the test editor or the SAP Protocol Data view to create or edit SAP get events and place verification points on them. When using the SAP Protocol Data view, you can select SAP screen elements from the screen capture to specify the SAP GUI identifier for the get event. Using this method to create or edit an SAP verification point is easier than adding it manually from the test editor.

The SAP Protocol Data view contains two pages that are synchronized with each other and with the test editor (Figure 4-17).

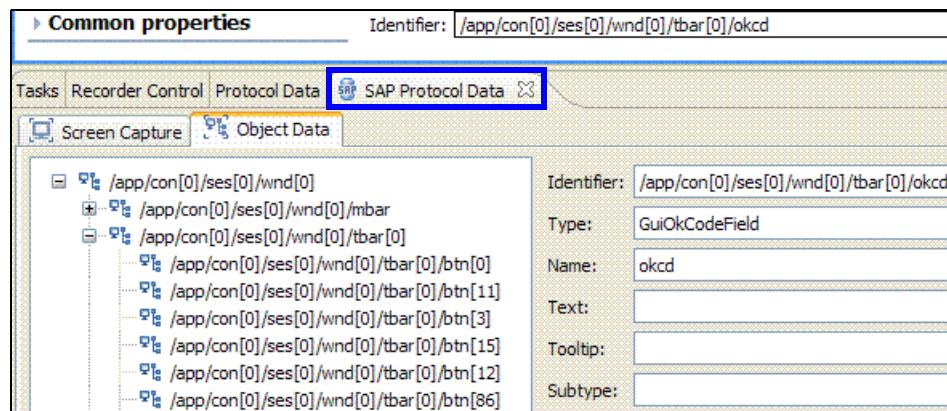


Figure 4-17 SAP Protocol Data view

- ▶ The *Screen Capture* tab displays a graphical screen capture of the SAP GUI. You can select all GUI objects such as windows, buttons, fields, or areas.

- ▶ The *Object Data* tab provides information about the selected GUI object: identifier, type, name, text, tooltip, and subtype. This information is used to determine possible events.

Viewing GUI data

The SAP Protocol Data view provides a graphical view of screens as they are displayed in the SAP GUI. In addition, it provides a view of the SAP GUI object data. The data displayed in the SAP Protocol Data view is synchronized with the test elements selected in the test editor.

To view test content in the SAP Protocol Data view:

- ▶ Open the test.
- ▶ Click the SAP Protocol Data tab to activate the view. If this tab is not open, then you can open this view by selecting *Window* → *Show View* → *Other* → *Test* → *SAP Protocol Data*.
- ▶ In the test editor, select the line corresponding to the transaction, screen, or SAP event that you want to view.
- ▶ In the SAP Protocol Data view, select the tab corresponding to the type of data or view of interest.
 - Clicking *Screen Capture* opens a page that displays the corresponding screen as recorded in the SAP GUI client. If an SAP event is selected, the corresponding field, button, or GUI object is highlighted in red.
 - Clicking *Object Data* opens a page that displays the SAP GUI object data for the corresponding object. This data can be used to identify the object in a test element.

SAP screen details

In the test editor, SAP screen elements are located in transactions and are the basic performance measurement unit for the test. These settings apply to the selected get event. Refer to Chapter 11, “Testing SAP enterprise applications” on page 383 for details on working with SAP screen elements.

SAP events

You can insert SAP GUI set or get events into your test to add items such as a field selection, a keyboard entry, or an interaction with the GUI. Get events are contained in SAP screen elements in the test suite. SAP screen elements can be windows, dialog boxes, or transaction screens that are part of a recorded transaction. You can only add set events. Verification points can only be added to get events.

SAP event details

In the test editor, SAP events are located in SAP screen elements and describe various recorded interactions with the SAP R/3® server. These details apply to both set and get events. Refer to Chapter 11, “Testing SAP enterprise applications” on page 383 for more on SAP event details.

Adding SAP set events

You can insert SAP GUI set events into your test to add items such as a field selection, a keyboard entry, or an interaction with the GUI. SAP set events represent user interactions with the SAP GUI, such as entering a value into a field. Set events are contained in SAP screen elements. SAP screen elements can be windows, dialog boxes or transaction screens that are part of a recorded transaction.

You can use either the test editor or the SAP Protocol Data view to create or edit SAP set events. When using the SAP Protocol Data view, you can select SAP screen elements from the screen capture and copy the information directly to the new event. Using the SAP Protocol Data view to create or edit an SAP event is much easier than adding an event manually from the test editor. After creating the event, you can use the test editor to easily change the value. You can also replace that value with a datapool variable or a reference.

Test editing that does not apply to SAP tests

Some of the editing and augmentation described in this chapter do not apply to SAP tests. The editing that does not apply is as follows:

- ▶ Transaction and screen names are generated automatically and are not editable. Therefore, renaming for readability does not apply (see 4.9, “Improving test reuse and documentation” on page 135).
- ▶ Content verification is not possible in SAP tests. See 4.6.2, “Verifying expected behavior in SAP tests” on page 96 for details on verification that you can do.
- ▶ Editing connection and timing parameters is more limited with SAP test than with HTTP tests. See “Editing SAP connections” on page 127 for information on what is possible.
- ▶ Digital certificates do not apply to SAP tests.

4.3.4 Editing Citrix tests

Citrix uses a proprietary protocol called the Independent Computing Architecture (ICA) Protocol, which is a much lower level protocol than HTTP or SAPGUI. Therefore, Citrix tests in Performance Tester are very different from HTTP tests, and editing them has a number of differences. Citrix tests are essentially a series of screens, then there is a sequence of attached events that include user events (mouse and keyboard inputs) and window events used for synchronization of playback.

All application logic executes on the server, and only screen updates, mouse movements, and keystrokes are transmitted through the ICA session to the server and client device. The only events the ICA protocol gives Performance Tester to use for synchronization are Window Create, Activate, and Destroy events. If an action such as clicking a link in a browser causes new data to be loaded into an existing window (such as a Web new page), Performance Tester has no way to know when that page is ready for the next click. It is therefore imperative that all such actions be preceded by an image synchronization event. Image synchronization events can be a bitmap hash comparison or textual utilizing the OCR technology. Image synchronizations should be inserted before the next action to ensure the application is ready.

After recording, you can edit the events in each window element. Because the recorded input is primarily made of low level keyboard and mouse input, you can streamline the test by, for example, replacing key-press events with string inputs. You can use the comments and recorded screen captures to make navigating through the test easier. You can replace recorded test values with variable test data, or add dynamic data to the test. You can also set verification points on window titles and coordinates or image synchronizations to validate that the test behaved as expected.

Within the test editor (see Figure 4-18), you can:

- ▶ Select values for datapools in the Test Data or Test Element Details (refer to 4.4, “Providing tests with test data” on page 64 for detailed steps).
- ▶ Select or modify correlated values in the Test Data or Test Element Details (refer to 4.5, “Manipulating data correlation” on page 81 for detailed steps).
- ▶ Add verification to all or part of the test by right-clicking in the Test Contents (refer to 4.6, “Adding verification to tests” on page 89 for detailed steps).
- ▶ Insert test elements by right-clicking in the Test Contents (refer to 4.7, “Adding test elements” on page 105 for detailed steps).
- ▶ Modify connection parameters in the Test Element Details (refer to 4.8, “Adjusting connections and timing” on page 120 for detailed steps).

- ▶ Edit page names and add comments for readability in the Test Contents and Test Element Details (refer to 4.9, “Improving test reuse and documentation” on page 135 for detailed steps).

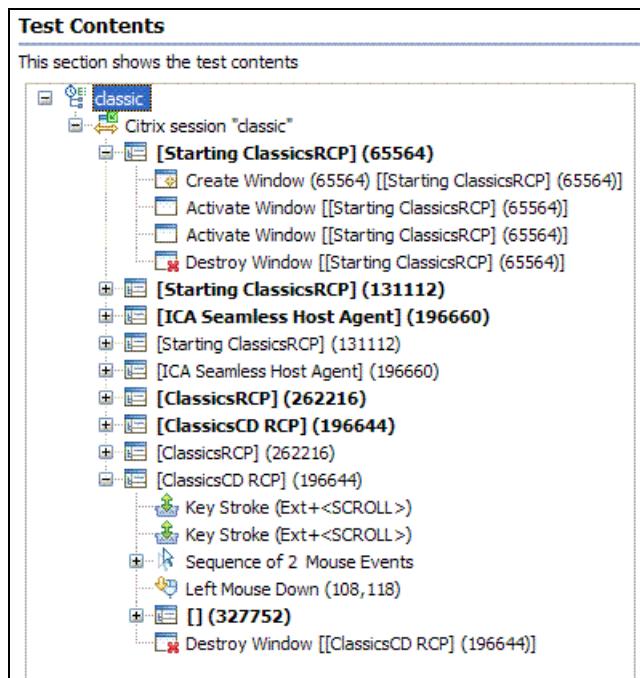


Figure 4-18 A Citrix test

Test editing that does not apply to Citrix tests

Some of the editing and augmentation described in this chapter do not apply to Citrix tests. The editing that does not apply is as follows:

- ▶ Most of the verification points, except for Window title verification points, do not apply to Citrix tests; refer to 4.6.3, “Verifying expected behavior in Citrix tests” on page 101.
- ▶ Some types of test elements such as loops are generally not added to Citrix tests (refer to 4.7, “Adding test elements” on page 105).
- ▶ Editing the timing in a Citrix test, such as adding delays, is generally not recommended because it can break the synchronization required for test execution (refer to 4.8, “Adjusting connections and timing” on page 120).
- ▶ Digital certificates do not apply to Citrix tests.

4.3.5 Other types of systems under test

Performance Tester can be used for other types of systems not explicitly listed as a supported test type because many systems use common protocols. Most ERP systems today use Web clients to some extent, some as the primary interface, and quite often complete performance testing can be done entirely by emulating communication through the Web interfaces. For example, you would performance test a PeopleSoft® deployment using HTTP tests in the same way that you would test any Web application.

4.4 Providing tests with test data

Perhaps the most common type of editing that you will perform is adding variable test data through the use of datapools. Using good sets of data is important to any kind of testing, but especially performance testing. It would be unrealistic, and could end in misleading test results, to use the same data every time for every simulated user during test execution. We therefore have to replace the initial data used for recording or test development with tables of data for execution. Performance Tester leverages the Eclipse Test and Performance Tools Platform (TPTP) datapool structures for handling of test data.

One common workflow for getting test data for tests in Performance Tester would be to:

- ▶ Determine and identify the data needed for testing.
- ▶ Get data from a database or spreadsheet, or generate the data.
- ▶ Export or save the data into CSV file format.
- ▶ Import the CSV files into Performance Tester datapools.
- ▶ Associate the datapools with one or more tests:
 - Configure the datapool for the intended execution.
 - Identify the test parameters and substitute with datapool values.

The following sections explain these steps in more detail.

4.4.1 Providing test data versus data correlation

In some cases you will provide the test data to input to a system during test execution. In other cases some data input to the system will come from the system itself. An example of this is a unique session ID or counter that is passed along with every request to the server, where the value must originate from the system itself. In this case you would not attempt to store the data in a table but instead would correlate the data after it is first received from the system.

When you want to replace data in a test with known values that can be saved in a table you can use datapools, for all other dynamic data you will use data correlation (see 4.5, “Manipulating data correlation” on page 81). Both cases involve the same kind of substitution in a test, the difference is where the data comes from.

4.4.2 Test data considerations

There are several considerations to how your tests will use test data that affects the datapool configuration, for example:

- ▶ Does each data record have to be unique for each simulated user?
- ▶ Is the sequence of test data important?
- ▶ Can data be repeated without resetting system data?
- ▶ Will multiple test machines be used for the same test runs using the same test data?
- ▶ How crucial is the speed of test execution?
- ▶ Does the test data contain secure confidential data, such as passwords?
- ▶ Will the test data set be very large (over 10,000 records)?

To address these considerations, you can set several options for attaching a datapool to a test. Note that these options are not tied to the datapool itself but only affect the association between the test and the datapool.

The options for the first five considerations are described in “Associating datapools with tests” on page 71. The issue of storing passwords is covered in “Adding authentication in HTTP tests” on page 131. The last consideration having to do with the size of the test data set is addressed in 4.4.8, “Custom test data handling” on page 79.

4.4.3 Datapool overview

Datapools are test artifacts managed in Performance Tester separate from tests, but they must be associated with one or more tests to be useful. The datapool file structure requires the Eclipse platform to read and edit, however data can be easily imported and exported. Comma-separated-value (CSV) files are a common format for data files, and most databases can export data into this format.

The Performance Tester test generator automatically identifies values in the recorded protocol that are likely to be variable parameters, such as fields from a user input form. These values are called *datapool candidates* and they are displayed in green by default in the Test Element Details area of the test editor (Figure 4-19). It is then up to the test developer to decide which datapool candidates will in fact be substituted with datapool variables in place of the recorded values. It is also up to the test developer to create or identify the datapool to use for the test.

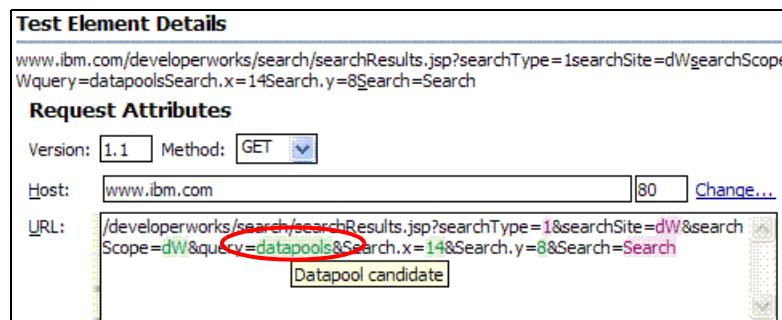


Figure 4-19 A URL encoded datapool candidate

Creating datapools is described in the subsequent section. Substituting datapool candidates with datapool values is described in 4.4.6, “Substituting test values with datapools” on page 74.

4.4.4 Creating datapools

Datapools can be created before or after the test that will use them is recorded. The same datapool can be used by more than one test, and a single test can use more than one datapool. The name and folder location of each datapool is important to help you better organize and reuse them.

Creating a blank datapool

The first procedure in this section describes the steps to create a new blank datapool. The procedure after this discusses initializing a datapool with imported data.

To create a new blank datapool:

- ▶ Select *File* → *New* → *Datapool*.
- ▶ In the New Datapool window, select the appropriate folder in the appropriate project (if more than one project is open) for the datapool.
- ▶ In the Name field, type the name of the datapool, then click *Next*.

- ▶ In the second New Datapool window, enter a description for the datapool. If you know the initial table dimensions you can enter them now, or add them later. Click *Next*.
- ▶ In the third New Datapool window, leave the fields alone and click *Finish*.

You now have a new blank datapool. The next step is to define the columns and enter data as described in “Editing datapools” on page 68.

Importing data into a datapool

Sometimes the test data that you require already exists in a database, spreadsheet, or other source. For this you can simply import the data into a datapool by first getting the data into a comma-separated-value (CSV) file. Most databases will have a utility to export data into this format, spreadsheet applications can save files into this format, and you can copy from documents or other files into this format using a word processor or text editor.

You might not already have existing test data, but instead plan on entering values during test development. In this case you still might want to consider entering the data into a spreadsheet first, especially if you will have more than a few dozen or so values. The reason is that spreadsheet applications, and even some text editors, are generally easier for data entry than the TPTP datapool editor.

Note: You can only import a CSV file when you first create the datapool; you cannot import data into an existing datapool.

To import the contents of a CSV file into a datapool:

- ▶ Select *File* → *New* → *Datapool*.
- ▶ In the New Datapool window, select the appropriate folder in the appropriate project (if more than one project is open) for the datapool.
- ▶ In the *name* field, type the name of the datapool, then click *Next*.
- ▶ In the second New Datapool window, enter a description for the datapool. Click *Next*.
- ▶ In the third New Datapool window, enter the path and file name or browse to the location of the CSV file (Figure 4-20).

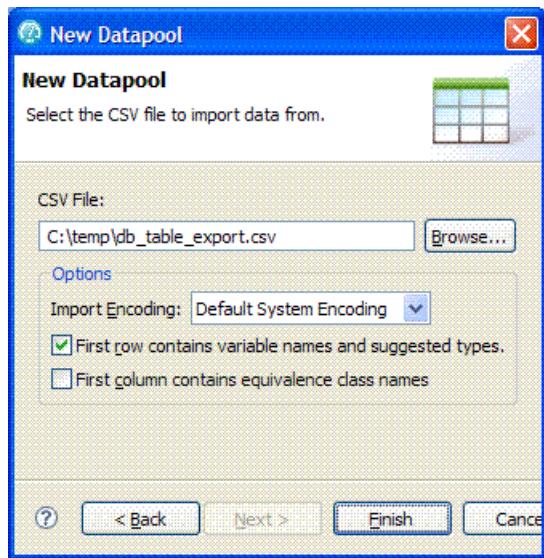


Figure 4-20 Importing data into a new datapool

- ▶ If the data in the CSV file is encoded differently from what the local computer expects, select the encoding from the Import Encoding list.
- ▶ If the first row of the CSV file contains column names, select *First row contains variable names and suggested types*. If this check box is not selected, columns are named Variable1, Variable2, and so on. You can change the names later when you edit the datapool.
- ▶ You will typically leave *First column contains equivalence class names* cleared. Equivalence classes are used in functional, rather than performance, testing. Select this check box only if you are using a datapool for both functional and performance testing, and the datapool already contains equivalence classes.
- ▶ Click *Finish*.

You now have a new datapool populated with test data from the CSV file.

Editing datapools

You can add, modify, or remove data from a datapool using the datapool editor, similar to the way you work with a spreadsheet. If your datapool changes are extensive, it might be easier to delete the datapool, enter the revised data into a CSV file, and import the data into a new datapool.

To enter data into or edit a datapool:

- ▶ Open the datapool to show the datapool editor which is on the equivalence class name tab (not the overview tab):
 - Click into the first row field and enter a value.
 - Press Enter to move the cursor to the next row.
 - Repeat as many times as needed.
- ▶ To modify existing data, select the cell and type over the data, just like you would do in a spreadsheet (Figure 4-21).

The screenshot shows a software interface titled "Datapool". At the top, there is a search bar labeled "SearchString:::::" and a dropdown menu with the word "datapool" selected. Below the search bar is a context menu with the following options:

- 0 datapool
- 1 Add Record...
- 2 Remove Record
- 3
- 4 Add Variable...
- 5 Remove Variable
- 6 Edit Variable...
- 7
- 8 Cut Ctrl+X
- 9 Copy Ctrl+C
- 10 Paste Ctrl+V

At the bottom of the menu, there is a link "Select Datapool in Test Navigator". The status bar at the bottom shows "Overview EquivalenceClass1".

Figure 4-21 Editing a datapool

- ▶ To add a new row, right-click on the row above the one to be added, and select *Add Record*.
- ▶ Alternately, if the last row in a datapool is selected then you can simply press Enter to add a new row.
- ▶ To remove a row, right-click the row to be deleted, and select *Remove Record*.
- ▶ To add a new variable (table column):
 - Right-click anywhere in the datapool cells, **not** on the variable names (column headers) at the top, and select *Add Variable*.
 - In the Add Variable window, enter a Name for the variable. The variable Type is optional.
 - In the Add drop-down list, select where you want the variable to be added relative to the other variables, then click *OK*.

- ▶ To change a variable (column) name:
 - Click on the variable name (column headers) at the top of the datapool editor.
 - In the Edit Variable window, modify the Name for the variable, then click *OK*.
- ▶ To change the order of a column:
 - Click on the variable name (column headers) at the top of the datapool editor.
 - In the Edit Variable window Move drop-down list, select where you want the variable to be moved relative to the other variables, then click *OK*.

Figure 4-22 shows the adding and editing of datapool variables.

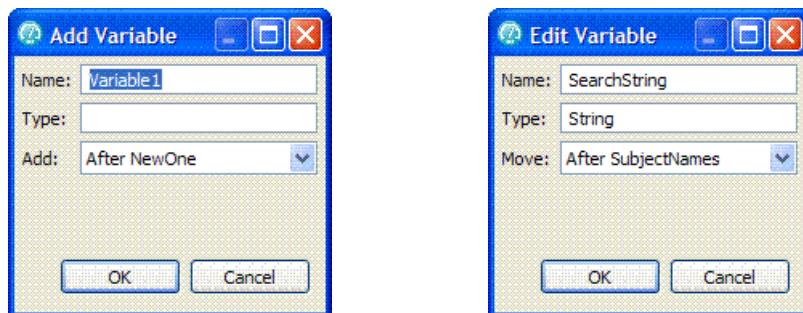


Figure 4-22 Adding and editing a datapool variable

Alternately, you can add, rename, or move variables from the Overview tab, in the Variables area.

- ▶ Save the changes.

Tip: For large data sets, you can maximize the datapool view in Performance Tester to see more rows and columns at once by double-clicking on the datapool tab (click on the name, not on the x, or you will close the datapool). To restore back to the original size, double-click on the datapool tab again or select the menu *Window → Reset Perspective*.

4.4.5 Adding datapools to tests

After you have a populated datapool, you can then use it in a test. To do this, you first associate the datapool with the test and configure options for how the test will use the datapool (see 4.4.2, “Test data considerations” on page 65). You then identify the datapool candidates in the test that you have to substitute with the datapool values.

Associating datapools with tests

Before a test can use variable data from a datapool, you must update the test to include a reference to that datapool.

To enable a test to use a datapool:

- ▶ Open the test that will use the datapool.
- ▶ Right-click the test name, and select *Add → Datapool*. The Import Datapool window opens, listing the datapools available to the test. If a datapool is already associated with that test, it will not appear in the list.
- ▶ Under the Matching resources list, select the name of the datapool that your test will use. When you select a datapool name, you see information about the datapool (Figure 4-23).

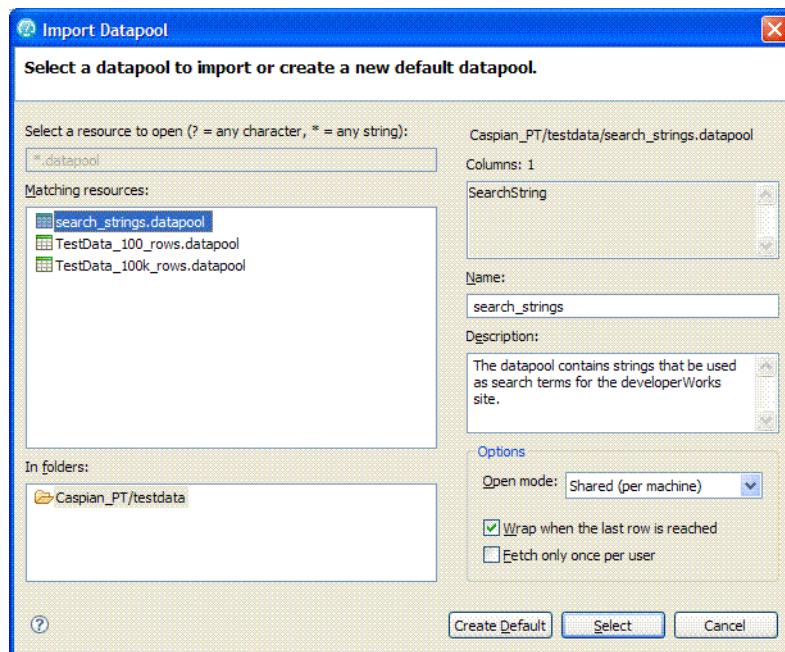


Figure 4-23 Attaching a datapool to a test

- ▶ Set the options which will affect how the test uses the datapool. See “Datapool options” on page 72 for a complete explanation of these.
- ▶ Click *Select*. A reference to the datapool is added to the test, and the Test Details area is updated with the datapool information.
- ▶ Save the test.

Tip: To see the datapools that are currently associated with a test, open the test and in the Test Contents area, click the first line of the test, which is the test name. Any datapools attached to the test will show in the Test Element Detail area, on the Common Options tab.

Now that you have created a reference between the test and the datapool, the next step is to associate a value in the test with a column in the datapool, described in 4.4.6, “Substituting test values with datapools” on page 741.

Datapool options

There are several options that affect how a datapool will behave with a test. These options can implement the factors described in 4.4.2, “Test data considerations” on page 65. These options are not saved as part of a datapool itself but instead only affect the association between the test and the datapool. The same datapool can be used with different options for a different test.

You typically set the datapool options when you attach it to a test, as described in “Associating datapools with tests” on page 71. You can also change the datapool options as the testing needs change.

To change the datapool options for a test:

- ▶ Open the test.
- ▶ In the Test Contents area, click the first line of the test, which is the test name. The datapools will show in the Test Element Detail area, on the Common Options tab.
- ▶ In the Datapools list, double-click on the name of the datapool you want to modify.
- ▶ In the Edit Datapool window, you can change the options, described in Table 4-1.
- ▶ Click *OK* to save the changes.

Table 4-1 Datapool options

| Option | Description |
|---|---|
| Open mode: Shared (per machine) (default) | <p>Virtual users on each computer draw from a shared view of the datapool, with datapool rows apportioned to them collectively in sequential order, on a first-come-first-served basis.</p> <p>This option makes it so that the virtual users or loop iterations will use data from different rows and that the server will see variable data. The exact row access order among all virtual users or iterations cannot be predicted, because this order depends on the test execution order and the duration of the test on each computer.</p> <p>If unique rows are needed for multiple test agent runs, this mode is not recommended, because each test agent gets its own datapool copy and re-uses all of the data rows in the datapool. Therefore, virtual users on different test agents will re-use the same data values.</p> |
| Open mode: Private | <p>Each virtual user draws from a private view of the datapool, with datapool rows apportioned to each user in sequential order.</p> <p>This option offers the fastest test execution and ensures that each virtual user gets the same data from the datapool in the same order. However, this option most likely results in different virtual users using the same row. The next row of the datapool is used only if you add the test that is using the datapool to a schedule loop with more than one iteration.</p> |
| Open mode: Segmented (per machine) | <p>Virtual users on each computer draw from a segmented view of the datapool, with data apportioned to them collectively from their segment in sequential order, on a first-come-first-served basis. The segments are computed based on how a schedule apportions virtual users among computers. For example, if a schedule assigns 25% of users to group 1 and 75% to group 2, and assigns these groups to computer 1 and computer 2, the computer 1 view will consist of the first 25% of datapool rows and the computer 2 view will consist of the remaining 75% of rows.</p> <p>This option prevents virtual users from selecting duplicate values (for example, account IDs). This mode should be used when the data rows can only be used once in the user scenario and you are using multiple test agents. If you disable wrapping, your tests will log an error and use a null value if the datapool rows for that test agent's datapool segment are exhausted. This will ensure that no row is used more than once.</p> <p>Note: This option requires the datapool to contain only one equivalence class. With the other options, equivalence classes are ignored.</p> |

| Option | Description |
|-----------------------------------|---|
| Wrap when the last row is reached | <p>This determines whether the test will reuse data when it reaches the end of the datapool.</p> <p>By default, when a test reaches the end of a datapool or datapool segment, it reuses the data from the beginning. To force a test to stop at the end of a datapool or segment, clear the check box beside Wrap when the last row is reached. Forcing a stop can be useful if, for example, a datapool contains 15 records, you run a test with 20 virtual users, and you do not want the last five users to reuse information. Although the test is marked Fail because of the forced stop, the performance data in the test is still valid. However, if it does not matter to your application if data is reused, the default of wrapping is more convenient. With wrapping, you need not ensure that your datapool is large enough when you change the workload by adding more users or increasing the iteration count in a loop.</p> |
| Fetch only once per user | <p>This determines whether the test will make the data in the datapool row the only value used by each virtual user.</p> <p>By default, one row is retrieved from the datapool for each execution of a test script, and the data in the datapool row is available to the test only for the duration of the test script. Select Fetch only once per user to specify that every access of the datapool from any test being run by a particular virtual user will always return the same row.</p> <p>This option is heavily used for login / password data as well as for digital certificate retrieval.</p> |

4.4.6 Substituting test values with datapools

After you have a datapool attached to your test, you can associate specific values in the test with datapool columns.

Note: To determine which datapool candidates to substitute with datapools, you must review the test in the test editor and locate the candidates in the Test Element Details. More importantly, you should have a good understanding of how the system under test receives and uses the variable input data. Without this knowledge, you risk substituting invalid values or missing critical parameters with uniqueness requirements

To associate a value in a test with a datapool column:

- ▶ Open the test.
- ▶ Locate and select a page or request containing a value that you want to replace with variable data.

Selecting a test page shows you a table that lists any datapool candidates and correlated data in that page. (If correlated data is not displayed, you can right-click the table and select *Show References*.) References are shown in blue letters, and datapool candidates are shown in black letters (Figure 4-24).

| Test Data | | | |
|-----------|-------------|------------------|---------|
| Name | Value | Substituted with | Owner |
| searchFor | doe%2C+john | | w3ib... |
| | | | |
| | | | |

Figure 4-24 Test data for a page showing datapool candidates

If the content of the Value column corresponds exactly with column data in your datapool, click the row and then click Datapool Variable. The Select datapool column window opens. Skip to step 6. You can ignore step 8, because URL encoding is preselected.

Otherwise, double-click the row to navigate to the page request containing the value that you want to replace from a datapool, and continue to the next step.

The value that you want to replace from a datapool might not be listed in any page table. In this case, manually locate the request string that includes the value.

- ▶ If the value that you want to replace from a datapool is part of a string that has been designated a datapool candidate, you must remove the light green highlight: right-click and select *Clear Reference*. For example, if you searched for **doe, john** in your test, the datapool candidate in your test is displayed as **doe%2C+john**. Suppose that you do not want to associate this candidate with a single datapool column containing data in the format **doe, john**. Instead, you want to associate **doe** and **john** with separate datapool columns. In this case, you must first remove the light green highlight.
- ▶ Highlight the value: With the left button pressed, drag your mouse over it.
- ▶ Right-click the highlighted value and select *Substitute from → Datapool Variable*. The Select datapool column window opens (Figure 4-25).

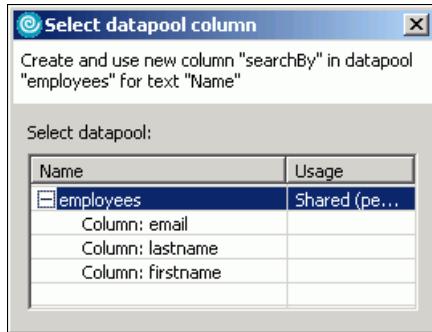


Figure 4-25 Selecting a datapool column for substitution

To use a datapool that is not listed, click *Add Datapool*: the Import Datapool window opens.

- ▶ Select the name of the datapool column that you want to associate with the test value.
- ▶ Click *Use Column*. To indicate that the association has been set, the highlighting for the selected test value turns dark green, and the datapool table for this page is updated as shown in Figure 4-26.

| Test Data | | | | |
|-----------|-------|------------------------|---------|--|
| Name | Value | Substituted with | Owner | |
| doe%2C+ | john | "firstname" variabl... | w3ib... | |
| searchFor | doe | "lastname" variabl... | w3ib... | |

Figure 4-26 Test data for a page showing datapool substitution

- ▶ **Optional:** Encode variable data when it is substituted from a datapool.
If a test value contains special characters such as spaces or commas, select the row and select *URL Encode*. With this option, special characters are encoded when variable data is substituted from a datapool. For example, data that is space-separated in a datapool column (such as **John Doe**) is substituted as **John%20Doe**. If URL encoding is not selected, the variable data that is substituted is literal. Do not enable URL encoding for datapools that contain data that is already encoded.
- ▶ Save the test.

4.4.7 Summary of datapool and test editing

This section describes a number of steps that are useful when editing datapools and tests. You can enlarge the test and the datapool, list the datapools used by a test, navigate from a row in a datapool to the corresponding element in the test, see the data for a page or request, and add or remove datapool references.

To see the datapool(s) used by a test:

- ▶ Open the test.
- ▶ In the Test Contents area, select the first line of the test.
- ▶ Under the Test Elements Detail area, Common Option, the Datapools table displays all datapools associated with the test (Figure 4-27).

The screenshot shows the 'Performance Test - dW_search' interface. On the left, under 'Test Contents', there is a tree view of the test structure. A red circle highlights the root node 'dW_search'. Below it, a red arrow points to the text 'Select test name'. Another red arrow points to the text 'Datapools listed here'. On the right, under 'Test Element Details', there is a 'Common Options' tab selected. A red box highlights the 'Datapools' table. The table has columns for Name, Usage, and Location. One row is visible: 'search_strings' (Usage: Shared (per machine), Location: /Caspian_PT/testdata/search). At the bottom right of the table is a blue button labeled 'Add Data'.

Figure 4-27 Datapools used by a test

To navigate from a row in a datapool to its corresponding (substituted) test element(s):

- ▶ Open the test and select the test name to display the attached datapool(s).
- ▶ Under the Test Elements Detail area, in the Datapools table, expand the datapool to display the datapool columns.
- ▶ Expand the datapool variable (column) you are interested in to display all substitution locations in the test.
- ▶ You can double-click on an expanded location row to jump to the place in the test contents where the datapool substitution is made (Figure 4-28).

| Test Element Details | | | | | | | | | | | | | | | | | |
|---|------------|--|------|-------|----------|----------------|------------|--|------------------------|-----------|--|---------------------------------|--|--|-----------------------------|--|--|
| dw_search | | | | | | | | | | | | | | | | | |
| Common Options HTTP Options | | | | | | | | | | | | | | | | | |
| Datapools | | | | | | | | | | | | | | | | | |
| <table border="1"> <thead> <tr> <th>Name</th> <th>Usage</th> <th>Location</th> </tr> </thead> <tbody> <tr> <td>search_strings</td> <td>Shared ...</td> <td>/Caspian_PT/testdata/search_strings.datapool</td> </tr> <tr> <td> └ Column: searchString</td> <td>Used by 2</td> <td>www.ibm.com/developerworks/search/search... IBM developerWorks &gt; Search results - sort by...</td> </tr> <tr> <td> Substituter: "datapools" - d...</td> <td></td> <td></td> </tr> <tr> <td> Custom code: custom.LogInfo</td> <td></td> <td></td> </tr> </tbody> </table> | | | Name | Usage | Location | search_strings | Shared ... | /Caspian_PT/testdata/search_strings.datapool | └ Column: searchString | Used by 2 | www.ibm.com/developerworks/search/search... IBM developerWorks > Search results - sort by... | Substituter: "datapools" - d... | | | Custom code: custom.LogInfo | | |
| Name | Usage | Location | | | | | | | | | | | | | | | |
| search_strings | Shared ... | /Caspian_PT/testdata/search_strings.datapool | | | | | | | | | | | | | | | |
| └ Column: searchString | Used by 2 | www.ibm.com/developerworks/search/search... IBM developerWorks > Search results - sort by... | | | | | | | | | | | | | | | |
| Substituter: "datapools" - d... | | | | | | | | | | | | | | | | | |
| Custom code: custom.LogInfo | | | | | | | | | | | | | | | | | |

Figure 4-28 Locations of all datapool substitutions

To see the substituted data for a request or page:

- ▶ Open the test.
- ▶ In the Test Contents area, expand the test and select the request or page you are interested in.
- ▶ Under the Test Elements Detail area, in the Test Data table, any value that is substituted with a datapool will display in green text show the datapool name (Figure 4-29).

| Test Element Details | | | | | | | | | | | | | | | | | | | | | | |
|---|-----------|---|-----------------------|-------|------------------|-------|-------|-----------|---|-----------------------|-------------|----|--|-----------------------|----------|----|--|-----------------------|----------|---|--|-----------------------|
| IBM developerWorks > Search results | | | | | | | | | | | | | | | | | | | | | | |
| Page title: IBM developerWorks > Search results | | | | | | | | | | | | | | | | | | | | | | |
| Primary request: www-128.ibm.com/developerworks/search/searchResults.jsp?searchType=1searchSite=dWsearchSc | | | | | | | | | | | | | | | | | | | | | | |
| Think time: 11236 milliseconds | | | | | | | | | | | | | | | | | | | | | | |
| Test Data | | | | | | | | | | | | | | | | | | | | | | |
| <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Substituted with</th> <th>Owner</th> </tr> </thead> <tbody> <tr> <td>query</td> <td>datapools</td> <td>"SearchString" variable, of search_strings datapool</td> <td>www.ibm.com/developer</td> </tr> <tr> <td>searchScope</td> <td>dW</td> <td></td> <td>www.ibm.com/developer</td> </tr> <tr> <td>Search.x</td> <td>14</td> <td></td> <td>www.ibm.com/developer</td> </tr> <tr> <td>Search.y</td> <td>8</td> <td></td> <td>www.ibm.com/developer</td> </tr> </tbody> </table> | | | Name | Value | Substituted with | Owner | query | datapools | "SearchString" variable, of search_strings datapool | www.ibm.com/developer | searchScope | dW | | www.ibm.com/developer | Search.x | 14 | | www.ibm.com/developer | Search.y | 8 | | www.ibm.com/developer |
| Name | Value | Substituted with | Owner | | | | | | | | | | | | | | | | | | | |
| query | datapools | "SearchString" variable, of search_strings datapool | www.ibm.com/developer | | | | | | | | | | | | | | | | | | | |
| searchScope | dW | | www.ibm.com/developer | | | | | | | | | | | | | | | | | | | |
| Search.x | 14 | | www.ibm.com/developer | | | | | | | | | | | | | | | | | | | |
| Search.y | 8 | | www.ibm.com/developer | | | | | | | | | | | | | | | | | | | |

Figure 4-29 Viewing datapool substitutions

To add a reference to (substitute a value with) a datapool:

- ▶ Open the test and go to the page or request which has the value to substitute.
- ▶ In the Test Element Details area, drag your cursor over the candidate, right-click, and select *Substitute From → Datapool Variable*. This is described in detail in 4.4.6, “Substituting test values with datapools” on page 74.

To remove a reference to a datapool:

- ▶ Open the test and go to the page or request with the datapool reference (where the value is substituted).
- ▶ In the Test Element Details area, place your cursor in the reference, right-click, and select *Remove Substitution* (Figure 4-30).

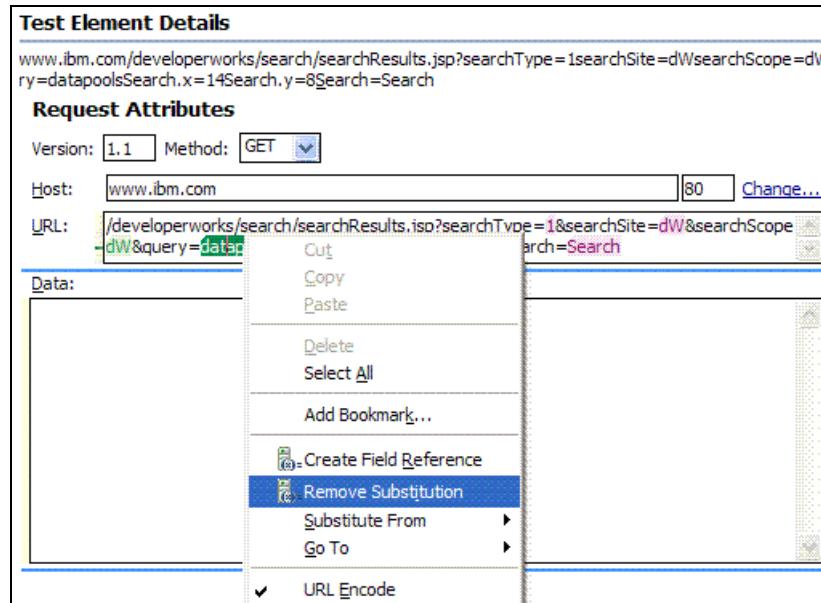


Figure 4-30 Removing a datapool substitution

4.4.8 Custom test data handling

Although datapools are the easiest and most common way to provide variable test data, they can become inefficient with very large sizes. This can result in long delays when starting up a test for execution. For performance testing of large systems it is not inconceivable to have test data requirements of one million records or more and in these scenarios the TPTP datapools might not be the most effective solution. However, Performance Tester does not limit you to using just datapools for test data; you can also use test data files directly through custom coding to handle whatever volume of data is required for your testing. This approach is described in the developerWorks® article *Handling test data with Rational Performance Tester 7.0: Part 2*, which can be found at:

http://www.ibm.com/developerworks/national/library/07/0130_davis2/

For more information on custom coding, see Chapter 12, “Testing Citrix application environments” on page 401.

4.4.9 Using datapools for digital certificates

Digital certificates are used for secure sockets layer (SSL) for many Web-based applications. When multiple users access such a system, each will have a unique client-side digital certificate containing an encryption key used for secure authentication. For Performance Tester HTTP tests, a recorded test will only have one digital certificate from the recorded session. You can then create a digital certificate store to hold multiple client certificates for multiple virtual testers to accurately load a system during test playback. See “HTTP Digital certificate overview” on page 131 for more information.

Not every Web application that utilizes SSL or has an HTTPS URL necessarily requires a digital certificate store for performance testing.

The digital certificates in a store are used by HTTP tests through datapools. The basic steps to do this are as follows:

- ▶ Create a digital certificate store, as described in “Creating a digital certificate store” on page 132.
- ▶ Create a datapool to link to the digital certificates.
- ▶ Associate the datapool with a test, as described in 4.4.6, “Substituting test values with datapools” on page 74.

Associating a digital certificate store with a datapool

You can associate the certificates in one or more certificate stores with a datapool to use multiple digital certificates during testing.

To use a digital certificate store with a datapool:

- ▶ Open a test for editing. On the *Common Options* page, click *Add Datapool*.
- ▶ Create a datapool with one column that contains a list of the certificates in the certificate store. See 4.4.4, “Creating datapools” on page 66 for more information on creating a datapools.
- ▶ Select *Fetch only once per user*.
- ▶ Save the datapool.
- ▶ On the *Common Options* page, under Digital Certificates, click *Add*.
- ▶ Select the certificate store that you created previously.

Note: The passwords for all of the certificates that you want to use for playback must be default.

- ▶ Type a Certificate Name for the digital certificate. Highlight this name and then click *Substitute from datapool*. Choose the datapool that you added previously, and then choose the column with the certificate name.
- ▶ Save the test.

When you run this schedule, the certificates from the certificate store will be submitted to the server.

4.5 Manipulating data correlation

Data correlation refers to replacing a value sent to the system under test with another dynamic value. This differs from using test data in datapools, where the values are static and can be saved in tables (see “Providing tests with test data” on page 64). When you want to replace data in a test with values that cannot be saved in tables then you will use data correlation. This uses the same kind of substitution in a test, the only difference is where the data comes from.

The substituted value typically comes from the system under test itself. For example, a system might generate a unique session ID value that is sent to the client; in this case, a Performance Tester test, and that value would then have to be used in all subsequent requests made during that session. For the test to work, it would have to correlate the session ID value from the server and replace it into every request transaction made in the test.

Instead of coming from a server, the replacement value could also come from some calculation or algorithm. For example, an application might need a unique incremented value for every transaction made in a given session but the value could be a simple counter plus some prefix string. In this case, the test could correlate the value from a simple custom code addition (see 4.7.6, “Custom code” on page 116).

4.5.1 Automatic correlation

A lot of required data correlation for HTTP and Siebel tests is done automatically by the Performance Tester. For example, HTTP URL path names by default are correlated if redirected by response codes such as 302 (found, redirecting). The generated test will automatically substitute values required for correct playback (Figure 4-31).

This capability can be modified or turned off or from *Window → Preferences, Test → Performance Test Generation → Data Correlation* tab (Figure 4-32). Other than changes these preferences, generally you should not have to modify the automatic substitution.

| Test Element Details | | | |
|---|--|--|---|
| developerWorks : Sign in | | | |
| Page title: | developerWorks : Sign in | | |
| Primary request: | www-128.ibm.com/developerworks/WiWeb/jsp/WSLogin.jsp | | |
| Think time: | 1542 | milliseconds | <input type="button" value="▼"/> |
| Test Data | | | |
| Name | Value | Substituted with | Owner |
| /developer... /i /i /i /us lang d d m | /developerworks/WiWeb/jsp /i /i /i /us en_US http://www-130.ibm.com... http://www-130.ibm.com... loginpage | (i)= "/developerworks/WiWeb/jsp" ... (i)= "i" - Response header (i)= "i" - Response header (i)= "i" - Response header (i)= "/us" - Response header (i)= "en_US" - Response header (i)= "http://www-130.ibm.com/de... (i)= "http://www-130.ibm.com/de... (i)= "loginpage" - Content | www-128.ibm.com/developer... www.ibm.com/i/v14/t/zz/... www.ibm.com/i/v14/button... www.ibm.com/i/v14/rules/... www.ibm.com/us/ www-128.ibm.com/develop... www-128.ibm.com/develop... www-128.ibm.com/develop... www-128.ibm.com/develop... |

Figure 4-31 An HTTP test with automatically correlated values

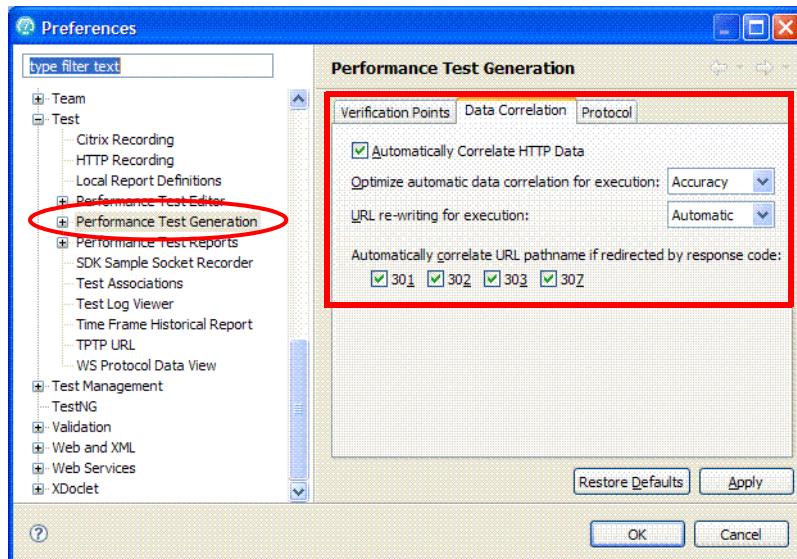


Figure 4-32 HTTP data correlation preferences

4.5.2 Manual correlation

There are a number of situations where you will manually configure data substitution. It is possible that a system might have an atypical session handling or other communication scheme where Performance Tester will not automatically correlate values in a way required to play back tests. Despite all of the automatic correlation, it is not rare that you might have to manually correlate some values.

Manually correlating data involves identifying a request parameter that has to be changed from the recorded *hard coded* value, then determining what the value needs to be substituted with and where that value is received in the test. When you know where the data will come from (in a server response), you can create a reference for it, then substitute the recorded request value with the reference.

Creating a reference

When you designate a test value as a reference, or a range of test data as a field reference, you can use the data elsewhere in the test. This is similar to creating a variable from a value in software code. A reference, usually located in response data, points to a specific value that you want to use from a subsequent test location, usually a request. You can also use a reference as input to an IF-THEN condition (see 4.7.4, “Conditions” on page 111) or as input to custom Java code that your test calls (see “4.7.6, “Custom code” on page 116).

A field reference points to a range of test data. Like a single reference you can use a field reference as input to an IF-THEN condition or custom code.

To create a reference or field reference:

- ▶ Open the test.
- ▶ Locate the value or range that you want to designate as a reference or field reference. You can create references and field references in these fields:
 - A request URL or post data (URL and Data fields)
 - A request or response header value (Value column of a Request Headers table or a Response Headers table)
 - Response content (Content field)

Note: The Test Search function is a useful way to locate values you want to reference, especially in response content. See 4.2.2, “Test search and replace” on page 46 for more information.

- ▶ To create the reference:
 - Highlight the value: With the left mouse button pressed, drag your mouse over the value.
 - Right-click and then select *Create Reference*. The value is highlighted in light blue to indicate that it is an unused reference. When you use it, the highlight changes to dark blue (Figure 4-33).

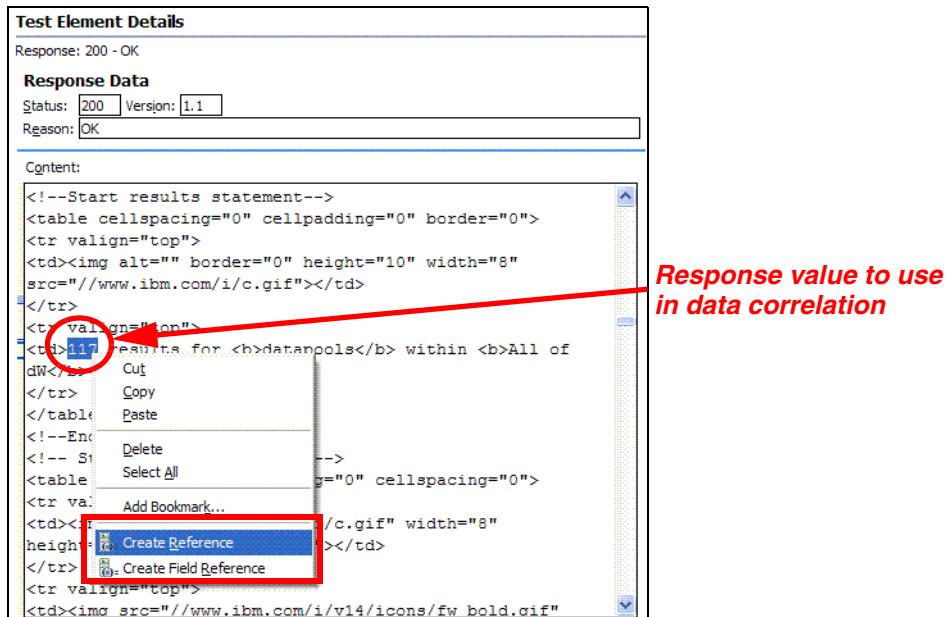


Figure 4-33 Creating a reference from a response value

- ▶ To create a field reference, right-click and select *Create Field Reference*. The field is highlighted in yellow to indicate that it is a field reference.

Correlating a request

If a test runs without error but does not get the results that you expected, you might need to correlate a value in a request with a reference in a previous response. The value that you want to correlate a request value with must already be designated a test reference (see “Creating a reference” on page 83).

To correlate a request:

- ▶ Open the test.
- ▶ Locate the value that should be replaced by a reference.
- ▶ Highlight the value: With the left mouse button pressed, drag the mouse over it.

- Right-click the highlighted value and select *Substitute from* → *Reference*, and select the correct reference. The value is highlighted in dark green to indicate that it has been correlated, and the correlation is added to the Test data substitution table for this page (Figure 4-34).

| Name | Value | Substituted with | Owner |
|-----------------|-----------|-------------------------|--------|
| perPage | 20 | ⇒ "20" - Content | www... |
| rankprofile | 8 | ⇒ "8" - Content | www... |
| type | all | ⇒ "all" - Content | www... |
| encodedQuery | datapools | ⇒ "datapools" - Content | www... |
| lastUserQuery1 | datapools | ⇒ "datapools" - Content | www... |
| searchSite | dW | ⇒ "dW" - Content | www... |
| searchScope | dW | ⇒ "dW" - Content | www... |
| displaySearc... | dW | ⇒ "dW" - Content | www... |
| summaryShow | on | ⇒ "on" - Content | www... |
| langEncoding | UTF8 | ⇒ "UTF8" - Content | www... |

URL Encoded
Datapool Variable...
Built-in Datasources...
Substitute From

Page Title Verification
Enable verification point

Expected page title:

⇒ "117" - Content
⇒ "8" - Content
⇒ "UTF8" - Content
⇒ "" - Content
⇒ "/I" - Response header
Find More...

Figure 4-34 Substituting a request value with a referenced response value

4.5.3 Ensuring proper data correlation

In this section we discuss various strategies to ensure data correlation.

General strategies for data correlation

In some cases, proper data correlation might be both complex and difficult but also essential for correct test execution. The following strategies and activities can help you successfully correlate data for your tests.

Understand the system under test and protocol

The better understanding you have of what gets passed back and forth between the servers and clients, the easier it will be to identify which values have to be substituted and from where. This applies both to general protocols, such as HTTP, as well as a specific system's design and implementation using the protocol. Without this knowledge, there is a higher risk of error in correlated values.

Search the test

When a particular request parameter is manually substituted, that same parameter will often be sent several more times later in the test sequence. For example, you can manually correlate a special session value on the second page of a test and that same value can also be sent on the 4th, 5th, and 8th page of the test. In these cases you would have to correlate every instance of the value or else the test would not execute correctly. For this reason, it is important to thoroughly search the test for the same or even similar parameters when manually substituting values. Refer to 4.2.2, “Test search and replace” on page 46 for more information on searching in a test.

Debug and validate test execution

Whenever you are manually correlating data in a test, there is some risk that an incorrect substitution could cause subtle or even hidden errors in test execution. If the errors are not detected, then the performance testing results could be invalid. This is why debugging and validating test execution is important when correlating data. See Chapter 5, “Test playback and debugging” on page 139 for information on playback and debugging tests.

Troubleshooting data correlation

When you run a test, you might notice that the server is not under the expected load or that your database is not being updated as expected. Incomplete or incorrect data correlation are likely causes of these problems.

The following procedure helps to identify data correlation problems early:

- ▶ Run a test individually or in a schedule with the Log Level for errors, failures, and warnings set to All.
- ▶ After the run, open the test log as explained in Viewing the test logs.
- ▶ As explained in the view, Inspecting test log details in the Protocol Data, verify that each call to the server returned the expected data.

The data correlation algorithms used during test generation are based on well-known best practices. However, because these practices are continually evolving, various types of errors can occur during automated data correlation:

- ▶ **Insufficient correlation:** Test values that should have been correlated were not. These are some possible causes:
 - Two parameters that should be correlated have different names.
 - A value should be correlated with a previous value that does not occur in the expected location.
 - A parameter or value should be correlated with a previous parameter or value that does not occur in the test because it is a computed value.

- ▶ **Superfluous correlation:** Unrelated test values were correlated.
- ▶ **Incorrect correlation:** Test values that should have been correlated were correlated incorrectly.

Insufficient correlation: Parameters have different names

When two parameters that should be correlated have different names, or occur in unexpected locations, automated data correlation fails to recognize that the two parameters are related. For example, consider the request:

`http://www.madeupsite.com?id=12345`

Suppose that this request should be correlated with the server response containing `customer_id=12345`, not `id=12345`. In this case, the `id` parameter has to be correlated with `customer_id`.

Data correlation typically relates a response value that was returned from the server with a subsequent request value. The automated correlation algorithms look in the usual places—URL and Post data—for correlation candidates. But other schemes for returning parameters are possible. For example, consider the request

`http://www.madeupsite.com?id=12345`

Suppose that this request should be correlated with the server response containing the name and entity pair `href name="id" entity="12345"`, not `id=12345`. In this case, the `id` parameter has to be correlated with `name="id"`, and value `12345` has to be correlated with `entity="12345"`.

Here are some additional causes of insufficient correlation:

- ▶ Siebel uses the star array format. Standard correlation algorithms do not understand how to retrieve from and substitute into this format.
- ▶ SOAP designates correlation parameters in external XML files. The correlation algorithms do not understand the correspondence between parameters in the external file and in the test.

To manually correlate data in these cases:

- ▶ In the test editor, use search or browse to locate the two parameters that should be correlated.
- ▶ Go to the parameter that occurs first in the test. If the parameter is not in a reference, put it in a reference, as explained in “Creating a reference” on page 83.
- ▶ Go to the second parameter. As explained in “Creating a reference” on page 83, correlate this parameter with the earlier reference.

Insufficient correlation: One parameter is unnamed

Sometimes a parameter or value needs to be correlated with a previous parameter or value that is not named in the test, because it is computed (for example, by a JavaScript™ program). In this case, in order to correctly correlate the data, you have to understand how and where the parameter or value was computed, and then use a custom code block.

For example, consider the Web address:

`http://www.madeupsite.com?login_stamp=12345_Dec_12_05`

The value for `login_timestamp` is the concatenation of the login ID and the current date. In this case, you require custom code that concatenates the login ID and the date.

For another example, suppose that the server returned the login ID and date as separate entities (`href "customer_id=12345" Date="Dec_12_05"`). In this case, you can put these parameters in separate references and, in subsequent requests using customer ID and date, substitute them separately.

Superfluous correlation

Automated data correlation is based on pattern matching: a parameter or parameter value is correlated with a subsequent parameter or parameter value with an exact or similar name. But sometimes parameters with exact or similar names are in fact unrelated. In the best case, unneeded correlation is either harmless or adds a slight load that is inappropriate. In the worst case, the application does not expect a correlation and fails during playback.

To remove a superfluous data correlation:

- ▶ In the test editor, use search or browse to locate the value that should not be correlated (blue letters indicate correlated data).
- ▶ Right-click the value and click *Remove Substitution*.

Here is an alternative:

- ▶ In the test editor, click a page that contains requests that should not be correlated.
- ▶ Right-click anywhere in the Test Data table and select *Show References*.
- ▶ Click a table row with the superfluous correlation (blue letters indicate correlated data) and select *Remove Substitution*.

Incorrect correlation

A parameter requiring data correlation might occur many times throughout a test. For example, a session ID parameter that is used initially when a user logs in might also be used in every subsequent request. If multiple instances of a parameter in a test are not same, the correlation algorithms might choose the wrong instance.

The Test Generation preferences include a preference named *Optimize automatic data correlation for*. These are the values you can select:

- ▶ **Accuracy:** Each occurrence of a parameter is correlated with the nearest previous occurrence. This is the default.
- ▶ **Efficiency:** Each occurrence of a parameter is correlated with a single previous occurrence.

Incorrect correlations are less likely—but are possible—with the Accuracy setting. To manually correct this problem:

- ▶ In the test editor, use search, browse, or the Test Data table for the page to locate the value that was incorrectly correlated.
- ▶ Right-click the value or table row and select *Remove Substitution*.
- ▶ Right-click the value again, select *Substitute from*, and select the correct parameter.

4.6 Adding verification to tests

An important condition to delivering sound performance testing results is being able to prove that the testing is valid. If there is any suspicion that tests are not realistically or accurately stressing or exercising a system, or if system behavior or functionality is questionable during testing or measurement, then there could be attempts to discredit the testing effort. It is therefore important to be able to validate and prove not only the performance testing results, but as many other aspects of system behavior as possible.

This can be done with various types of verification provided by Performance Tester. These features can verify a number of things, for example that the server functioned with acceptable non-error responses during test runs, or that the correct records were returned from a database under load.

Performance Tester provides different verification for different types of tests. While there are similarities between the verification in different tests, the following sections explain the purpose and capability of verification by the type of test.

Adding verification points during test generation

You can add verification to tests while editing them, or you can have the test generator automatically add verification when the tests are created. To enable automatic generation of verification points, select *Window* → *Preferences* → *Test* → *Performance Test Generation* and set the preferences. You can select automatic verification for each kind of verification for different types of tests.

4.6.1 Verifying expected behavior in HTTP and Siebel tests

Performance Tester provides the same verification for both HTTP and Siebel tests (Figure 4-35). The only difference is that there are additional content verification points available for specific Siebel response strings, as discussed in “Siebel response content” on page 96.

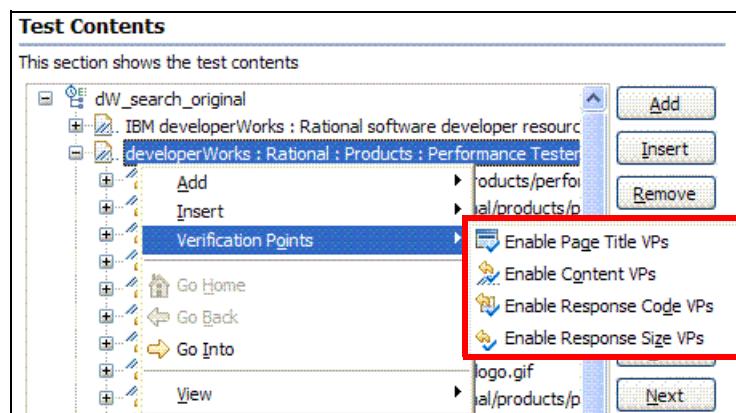


Figure 4-35 Verification points for HTTP and Siebel tests

Page titles

Page title verification points report an error if the title returned by the primary request for the page is not what was expected. Although the comparison is case-sensitive, it ignores multiple white-space characters (such as spaces, tabs, and carriage returns).

To specify the expected page title:

- ▶ Open the test.
- ▶ Right-click the test name or a page, and select *Enable Page Title VPs*. Your choice determines whether the verification point is added to all pages in the test or to one page.
- ▶ Click the page title to display the editing fields in the Test Element Details area.

- ▶ Ensure that the Expected page title field shows the string that you expect to be included in the response. Although you can change the string, the value listed is what was returned between the <title></title> tags during recording (Figure 4-36).

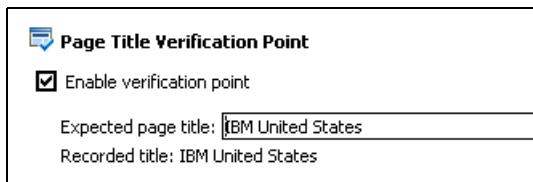


Figure 4-36 Page title verification point

Response codes

Response code verification points report an error if the returned response code is different from what you expected.

To specify the expected response code:

- ▶ Open the test.
- ▶ Right-click the test name, a test page, or a request, and select *Enable Response Code VPs*. Your choice determines whether a verification point is added to every request in the test, to every request in a page, or to a particular request (Figure 4-37).

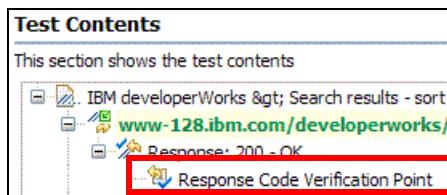


Figure 4-37 Response code verification point in test contents

- ▶ Select the verification point to display the response code editing fields in the Test Element Details area (Figure 4-38).

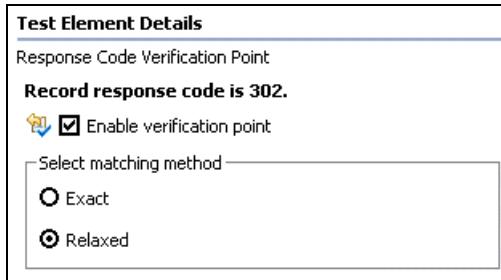


Figure 4-38 Response code verification point element details

- ▶ To disable an individual response code verification point, clear *Enable verification point*.
- ▶ From the Select matching method list, select an option to indicate how closely the response code that is actually returned during the run must match the recorded value:
 - If you click *Relaxed*, response codes that are in the same category (for example, 200, 201, 203, 209) are considered equivalent. An error is reported if the response code is not in the expected category.
 - If you select *Exact*, an error is reported if the response code does not match the specific expected value.

Response sizes

Response size verification points report an error if the size of a response is different from what you expected.

To specify the expected response size:

- ▶ Open the test.
- ▶ Right-click the test name, a test page, or a request, and select *Enable Response Size VPs*. Your choice determines whether the verification point is added to all test pages, to a page in the test, or to a particular request (Figure 4-39).

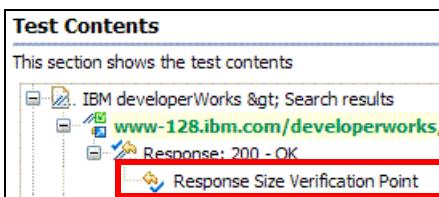


Figure 4-39 Response size verification point in test contents

- ▶ Select the verification point to display the response size editing fields in the Test Element Details area (Figure 4-40).

The screenshot shows the 'Test Element Details' dialog box with the following details:

- Section Header:** Response Size Verification Point
- Text:** Recorded response size is 206 bytes.
- Checkboxes:** A checked checkbox labeled "Enable verification point" is present.
- Section Header:** Select matching method
- Radio Buttons:**
 - Exact: bytes
 - At least:
 - At most:
 - Range (bytes):
 - Range (%):
- Note:** Verify that the response size is exactly 206 bytes.

Figure 4-40 Response size verification point element details

- ▶ To disable an individual response size verification point, clear the *Enable verification point* field.
- ▶ From the Select matching method list, select an option to indicate how closely the response size that is returned in the page request must match the recorded response size:
 - The default matching method for a primary request is *Range*, with a percentage adjustment based on the value in the field on the right. (You can change the percentage basis as well as the matching method.) When the test is run, an error is reported if the actual response size falls outside the adjusted range.
 - For other requests, the default matching method is *Exact*. When the test is run, an error is reported if the response size does not exactly match the expected size.

Response content

Content verification points report an error if a response does not contain an expected string. This type of verification can be crucial to verify functionality during load or stress testing where a system might continue to generate valid responses but with incorrect data.

This is also important because many Web applications today might have enough servers and load balancers to continue running under very high loads (that is, the server will not crash or produce error codes), but the system will instead respond with various messages such as *unable to process your request*. These unavailable messages have non-error server codes and might have similar sizes to the correct responses, making them difficult to distinguish from expected behavior.

You can add content verification points to an individual response, a page, or to the entire test. If you enable content verification for an entire test, the Create/Enable Content Verification Point window opens, which you can then use to select the text strings that you want to look for in multiple responses throughout the test. You can define a content verification point and then insert it in all or specified responses throughout the test or test page, by clicking *Skip*, *Enable All*, or *Enable* (Figure 4-41).

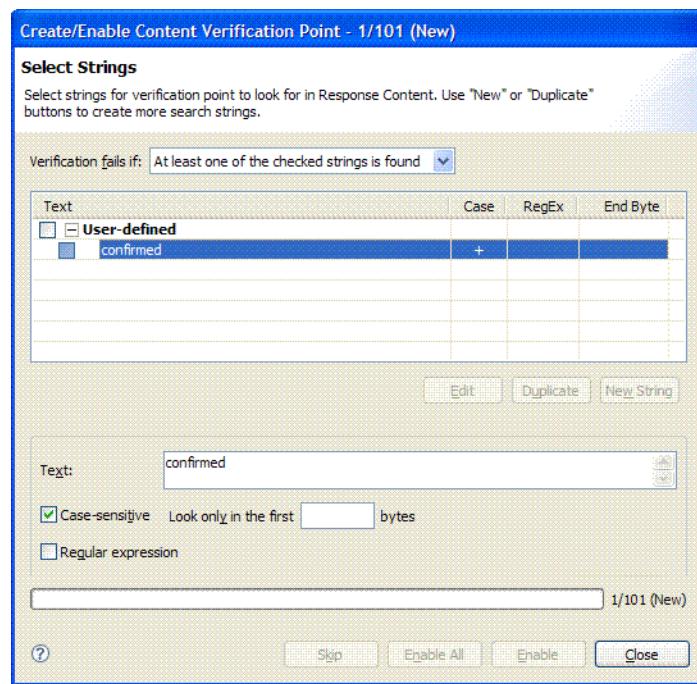


Figure 4-41 Creating response content verification points for a test

If you defined a content verification point for a particular response, this can then be added to other responses in the test without having to re-define the same content verification point.

For HTTP tests, only the User-defined category is displayed initially until you create your own verification strings or regular expressions. There are two basic ways to create a new verification point:

- ▶ To create a new string from scratch, click *New String*.
- ▶ To create a new string by editing another string, select it and click *Duplicate*.

If content verification was enabled for a single request or response, a content verification point is inserted in the specified response (Figure 4-42).



Figure 4-42 Response content verification point in test contents

You can edit a specific content verification point in Test Element Details that you inserted from the Create/Enable Content Verification Point window. In the following instructions, only the first four steps apply when you are editing in Test Element Details.

To define a content verification point, open the test, and follow these steps:

- ▶ Right-click the test name, a test page, or a request, and select *Enable Content VPs*. Your choice determines whether the verification point is added to all test pages, to a page in the test, or to a particular request.
- ▶ From the *Verification fails if* drop-down list, select either *At least one of the checked strings is found* or *None of the checked strings are found*.
- ▶ In the list of strings in the *Text* area, select the strings that the content verification point should search for. If the required string does not exist then you can add it by clicking *New String*.
- ▶ Click *Skip* to advance to the next response without inserting the content verification point in the current response. If the scope is the test, the initial response is the first response in the test. If the scope is a test page, the initial response is the first response in that page.
- ▶ Click *Enable All* to insert the verification point into every test response (if the scope is the test) or every page response (if the scope is a page in the test).
- ▶ Click *Enable* to insert the verification point into the current response.
- ▶ Click *Close* when done.

Siebel response content

Enabling content verification points in a Siebel test are identical to HTTP tests except that many common Siebel message strings are already defined. When you enable content verification in a Siebel test, the Create/Enable Content Verification Point window will initially contain strings that are of interest in Siebel applications. You can select any of these or create your own user defined strings (Figure 4-43).

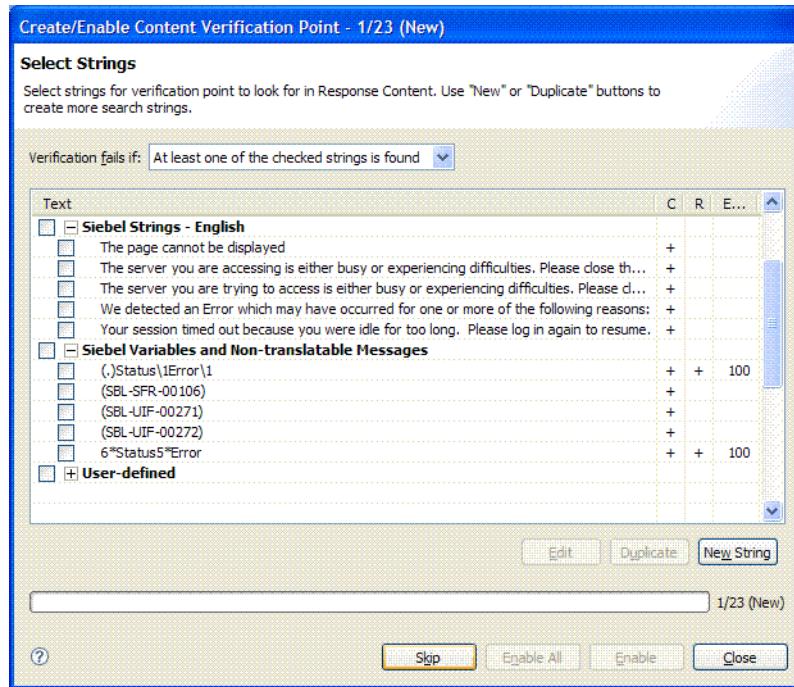


Figure 4-43 Siebel response content verification points

4.6.2 Verifying expected behavior in SAP tests

Performance Tester provides verification points for SAP tests, some of which are similar to those for HTTP. Two types are added from the Test Contents right-click menu (Figure 4-44), while the third is added from the SAP Protocol Data view. When using the SAP Protocol Data view, you can select SAP screen elements from the screen capture to specify the SAP GUI identifier for the get event. Using this method to create or edit an SAP verification point is easier than adding it manually from the test editor.

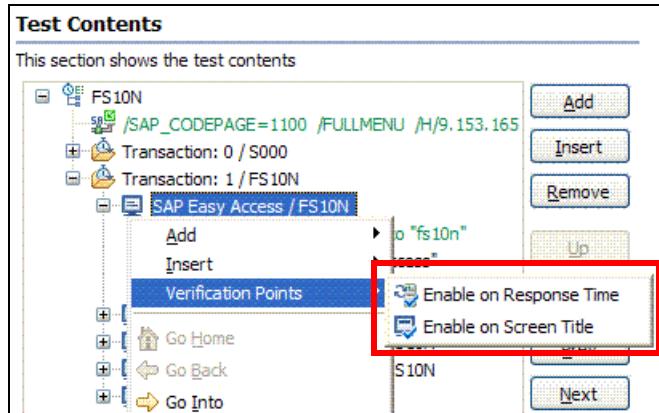


Figure 4-44 Verification points for SAP tests

Screen titles

Screen title verification points report an error if the title of an SAP screen is different from what you expected. To specify the expected screen title, perform the following steps:

- ▶ Select the SAP screen in the test editor and ensure that screen title verification is enabled for the SAP screen. The Test Element Details area includes a Screen Title Verification Point section:
 - If screen title verification was enabled for the entire test, *Enable verification point* is selected for all SAP screens in the test.
 - If screen title verification was enabled for a specific SAP screen, *Enable verification point* is selected for the selected SAP screen.

You can enable or disable screen title verification for a specific SAP screen in the test editor by selecting or clearing *Enable verification point*.

- ▶ Ensure that the Expected screen title field shows the string that you expect to be included in the page title that is returned when this page is loaded.

When the test was recorded, SAP returned a default title for this screen. This value is displayed in the Recorded title field, and is automatically copied to the Expected page title field when *Enable verification point* is selected. You can change the string in the Expected page title field as you like (Figure 4-45).

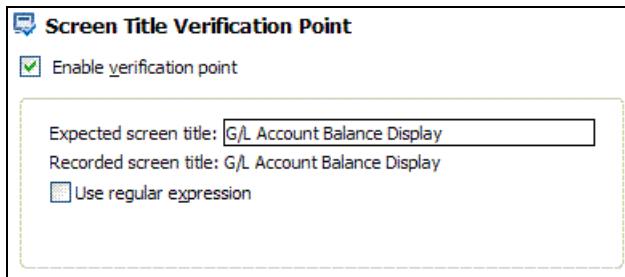


Figure 4-45 Screen title verification point

Whenever the test runs with page title verification enabled, an error is reported if the title returned for the page does not contain the expected title. Although the comparison is case-sensitive, it ignores multiple white-space characters (such as spaces, tabs, and carriage returns).

Response times

SAP request response times measure the delay between the moment the user submits a server request and the moment the server responds. Response time data is provided by the server. You can set a verification point on a response time threshold value. If the test encounters a response time above the threshold, the verification point is failed.

When the *Verification points for SAP request response time threshold* option is selected in the SAP Test Generation preferences, all SAP server request elements contain a response time verification point. The default threshold value is calculated by applying a multiplier to the recorded response time. You can change the default multiplier in the SAP Test Generation preferences. The response time measurements are displayed in the SAP server request element, which is the last element in an SAP screen.

To define a response time verification point, perform the following steps:

- ▶ In the test editor Test Contents area, select an SAP server request element inside an SAP screen element.
- ▶ In the Test Element Details, select *Enable verification point*, and then enter the Response time threshold in milliseconds (Figure 4-46). If the test encounters a response time that is higher than the threshold, the verification point is failed.

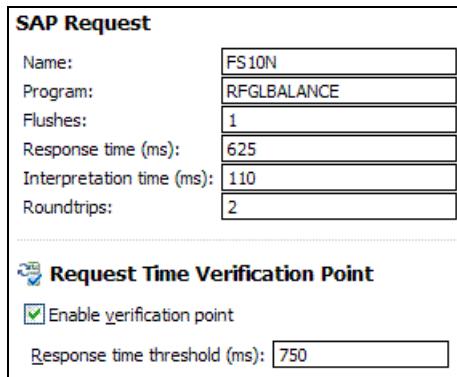


Figure 4-46 Request response time verification point element details

SAP verification points

SAP get elements enable you to retrieve a value from the SAP GUI to create verification points on an SAP screen element. This is somewhat similar to a content verification point for HTTP and Siebel tests.

When you add SAP verification points, SAP get elements retrieve the data from objects in the SAP GUI such as windows or text fields. SAP get elements are contained in SAP screens in the test suite. SAP screens can be windows, dialog boxes, or transaction screens that are part of a recorded transaction.

You can use either the test editor or the SAP Protocol Data view to create or edit SAP get events and place verification points on them. When using the SAP Protocol Data view, you can select SAP screen elements from the screen capture to specify the SAP GUI identifier for the get event. Using this method to create or edit an SAP verification point is easier than adding it manually from the test editor.

The SAP Protocol Data view contains two pages that are synchronized with each other and with the test editor:

- ▶ **Screen Capture** displays a graphical screen capture of the SAP GUI. You can select all GUI objects such as windows, buttons, fields or areas.
- ▶ **Object Data** provides information about the selected GUI object: identifier, type, name, text, tooltip, and subtype.

To add verification points to an SAP performance test, perform the following steps:

- ▶ Open the test editor and the SAP Protocol Data view. If the SAP Protocol Data view is not open, select *Window → Show View → Other → Test → SAP Protocol Data*.

- ▶ In the Test Contents area of the test editor, expand a transaction and an SAP screen. The SAP Protocol Data view displays a screen capture of the selected transaction.
- ▶ Inside the transaction, select the item for which you want to enter a new value. The Screen Capture page of the SAP Protocol Data view displays the screen capture of the SAP GUI with the corresponding GUI object highlighted in red.
- ▶ In the SAP Protocol Data view, right-click the GUI field where you want to enter a new value, and then select *Create Verification Point* (Figure 4-47). This opens the Create Verification Point wizard, which already contains the Identifier data from the recorded session.

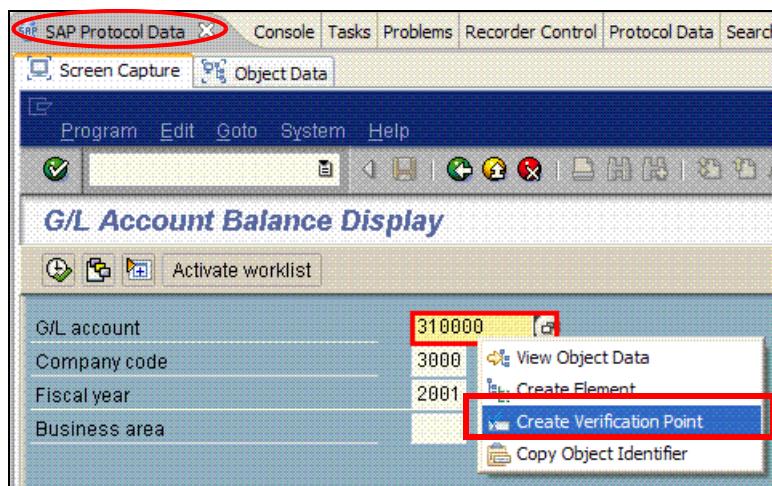


Figure 4-47 Selecting an object for an SAP verification point

- ▶ Specify the expected value for the verification point (Figure 4-48):
 - If you want to verify a text value in the SAP GUI object, ensure that *Verify text* is selected and type the Expected value that you want to verify. Click *Finish*.
 - If you want to verify advanced properties of the SAP GUI object, you can select *Advanced*, and then specify the properties attached to the GUI object as well as the Expected values. Refer to the SAP documentation for information about these properties.

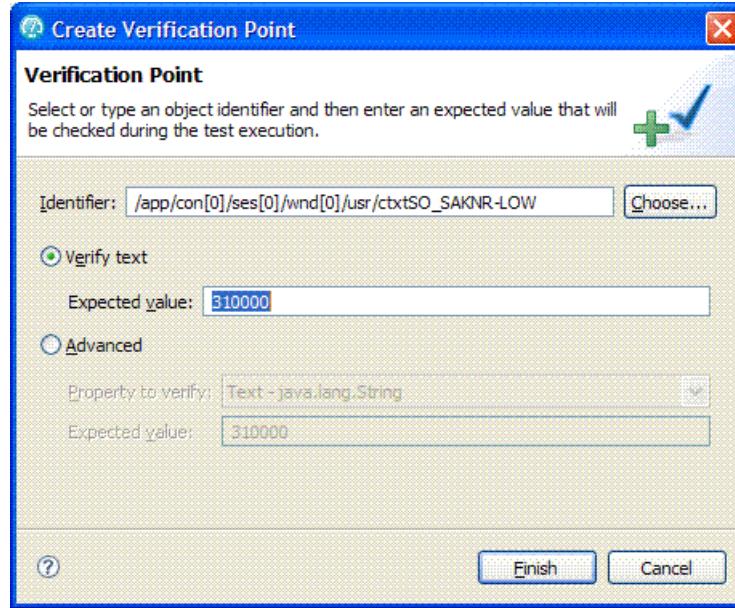


Figure 4-48 Creating an SAP verification point

After creating the event, you can use the test editor to easily change the value. You can also replace that value with a datapool variable or a reference. You can enable and disable SAP verification points in the test editor.

4.6.3 Verifying expected behavior in Citrix tests

Due to the nature of the Citrix protocol and the low level tests generated by Performance Tester, there is only one kind of verification point for Citrix tests (Figure 4-49). You can also measure response times in a Citrix test to verify behavior, although this is not a pass or fail type of verification. Each of these kinds of verification are described in the following sections.

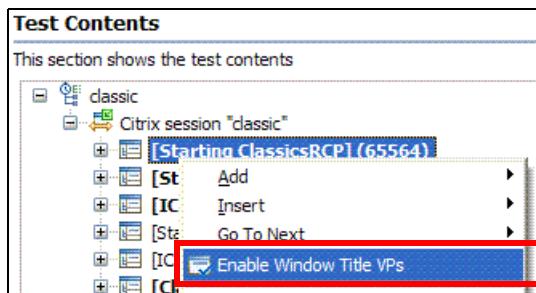


Figure 4-49 Verification point for Citrix tests

Window titles

Window title verification points produce a fail status in the test execution report if the window titles are different from what you expected. These are very similar to HTTP and Siebel page title and SAP screen title verification points.

To specify the expected window title:

- ▶ Ensure that window title verification is enabled for the selected window event. The Test Element Details area includes a Verification Point section:
 - If window title verification was enabled for the entire test, the *Enable verification point* check box is selected for all windows in the test.
 - If window title verification was enabled for a specific page, the *Enable verification point* check box is selected for the selected window.

To enable or disable window title verification for a specific window, go to that page and either select or clear the *Enable verification point* option.

- ▶ Ensure that the *Expected title* field shows the string that you expect to be included in the window title that is returned when this window is displayed (Figure 4-50).

When the test was recorded, the Citrix server returned a title for the current window. This value is displayed in the *Recorded title* field, and is automatically copied to the *Expected title* field when the *Enable verification point* check box is selected. You can change the string in the *Expected title* field as needed.

To use standard regular expressions in the *Expected title* field, select *Use regular expressions*.

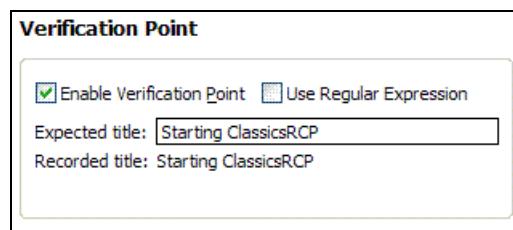


Figure 4-50 Window title verification point

Measured response times

A response time measures the delay between a specified start event and a specified stop event. Typically, the start of the measurement is set to a mouse or keyboard input and the stop is set to a window create, window activate, or window destroy event that is a response to the input event.

When the *Automatically Generate Response Time Measurements* option is selected in the Citrix Test Generation preferences, the recorded test already contains a series of response time measurements based on window creation events, and the user input events that occurred before the window creation. The response time measurements are displayed in the test element of the test editor.

To define a response time measurement:

- ▶ In the test editor Test Contents area, select a keyboard or mouse event that will start the response time measurement.
- ▶ Press the Ctrl key and select the window event that will stop the response time measurement. The two events are selected in the test.
- ▶ Right-click either the start or stop element, and select *Create Response Time*.
- ▶ A Create Response Time window displays information about the new response time measurement. If the new response time measurement replaces a previous one, click *Yes*. Otherwise, click *OK* (Figure 4-51).

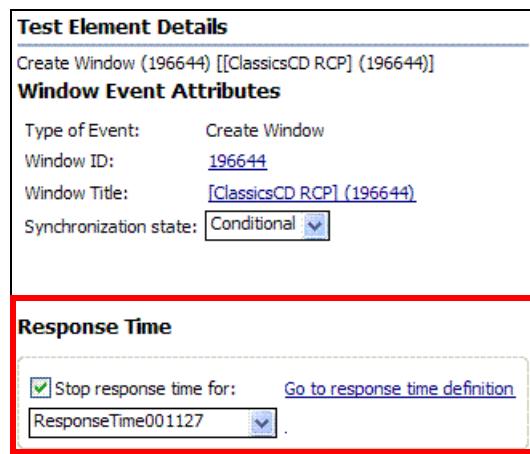


Figure 4-51 Response time in a Citrix test

You can view all the response times of a test by selecting the Citrix session element in the test. Response times are listed in the *Response Time Definitions* table where they can be renamed or deleted (Figure 4-52).

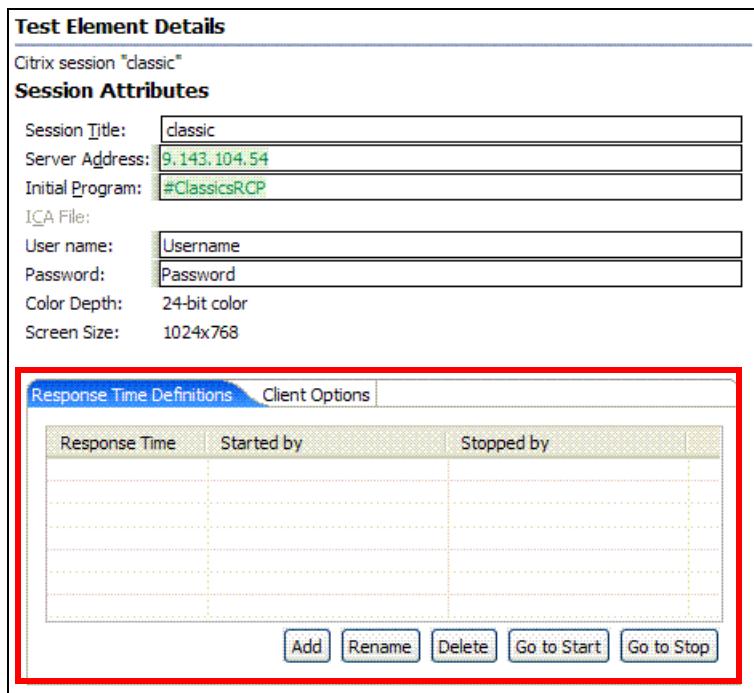


Figure 4-52 Response time definitions in a Citrix test

4.6.4 Custom verification

Beyond the automatic verification provided by Performance Tester described previously, you can also perform additional custom verification based on specific needs. This can be implemented through the use of custom code which is described in detail in Chapter 13, “Testing with custom Java code” on page 429. For example, you might need a test to perform some calculation with server returned values, then compare this against other data coming from the servers under test to validate correct system functionality under stress. To do this, you can pass server returned values to a custom code class, perform an operation or call a method in the custom code, then log a verdict (pass, fail, and so forth) based on comparisons or conditions.

This method for verifying system behavior can apply to all types of tests in Performance Tester. For a brief overview of adding custom code to a test, refer to “Creating and adding new custom code” on page 117. For more information on custom coding, see Chapter 13, “Testing with custom Java code” on page 429.

4.7 Adding test elements

Performance Tester provides a number of test elements that you can add to tests to extend the initial capability. The test elements allow you to model real user activity by emulating a user pausing to think or wait (delays), repeating certain steps (loops), making choices (conditions), or other transactional behavior. These elements add functionality to a test similar to adding lines of code to a program, however, test elements are added graphically.

4.7.1 Inserting and positioning elements

Performance Tester provides several buttons located in the center of the test editor, between the Test Contents and the Test Element Details, for editing elements in a test. These same functions can also be accessed from the right-click menu when selecting elements within the Test Contents. You must be aware of what item is selected in the Test Contents when using these buttons (Figure 4-53).

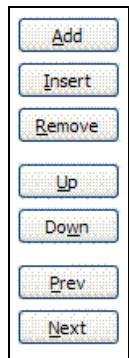


Figure 4-53 Test element editing buttons

Add and Insert

All test elements can be put into a test through either the *Add* or *Insert* function in Performance Tester. Adding an element adds a new child object to the selected element, which would place the new element *under and within* the selected element (Figure 4-54). When using the right-click menu to add, Performance Tester displays a black bar showing the location in the test where the new element will be added.

Inserting an element inserts a new child object before the selected element, which would place the new element *above* the selected element (Figure 4-55). When using the right-click menu to insert, Performance Tester displays a black bar showing the location in the test where the new element will be inserted.

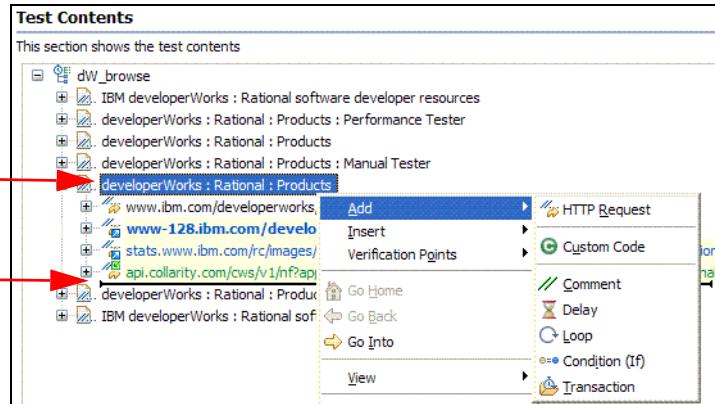


Figure 4-54 Adding an element

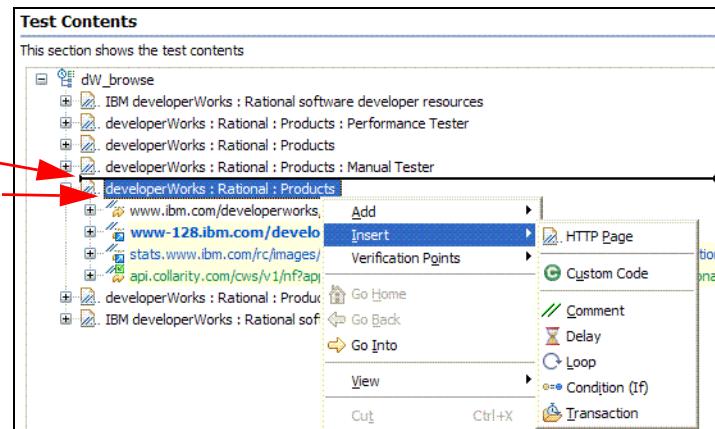


Figure 4-55 Inserting an element

Choosing whether to use Add or Insert depends somewhat on what you are putting into the test and where you want to put it. The choice is not critical, however, because you can move elements after they are added.

Moving elements up or down

You can reposition elements in a test by selecting them and using the *Up* and *Down* buttons. This changes the element's position in the test which affects the order in which it will be reached and executed. Note that elements can only be moved up and down within the same *level*, that is you cannot move an element down into a transaction if it was initially placed between transactions.

Removing elements

You can delete an element from a test by selecting it and clicking on the *Remove* button, which is the same as selecting *delete*. There is no *Undo* available when you remove an item, although this is considered an unsaved change that you could restore by exiting without saving (see “*Italics indicates changes not saved*” on page 44).

4.7.2 Delays

The delay element provides the simplest way to add timing control at a high level between transactions. This is similar to a simple sleep or delay function in software code. These can be used to emulate user delays without affecting a transaction, or as workarounds for synchronization issues (less common).

To add a  **Delay** to a test:

- ▶ Open the test.
- ▶ Select the element in the test just after where you want to add the delay.
- ▶ Right-click and select *Insert* → *Delay*.
- ▶ In the Test Element Details, select the unit of time (milliseconds, seconds, minutes, or hours) from the drop-down list.
- ▶ Enter a number for the delay duration.

Delays versus connection think times

In addition to the delay test element, you can also add simulated pauses by adjusting *think times*. Delays and think times are similar in that both create pauses between requests and responses, and neither are counted as time waiting for system responses. However, they are different in several ways and you should use the correct one accordingly.

The first primary difference is that think times can be varied for each simulated user so that each user plays back a slightly different simulated think time during test execution. Delays are always the same for every simulated user in a test. A second difference is that schedule settings can override the think time values in a test, which is typical when simulating workloads in Performance Tester. Delays are not affected by schedule settings. Finally, although think times are not counted as a system response time, they are considered part of a transaction. Delays occur *between* test elements.

You should use a delay when a test requires a pause of a specific time at a particular point in the test, similar to the way you would add a sleep or time delay in software code.

Delays in Citrix tests

Because of the nature of Citrix performance tests, it is important to preserve the synchronization of events. Adding delays to a Citrix performance test is likely to break the synchronization and produce execution errors. See “Timing in Citrix tests” on page 123 for more information.

4.7.3 Loops

A loop element allows you to repeat steps within a test. This is added so that other elements, typically requests and responses, are put inside the loop. This is similar to adding a simple for-next routine in procedural software code to repeat certain steps a number of times. Loops can also provide timing and rate control over transactions in a test.

To add a  Loop to a test:

- ▶ Open the test.
- ▶ Select the test items (pages, transactions, and so forth) that you want to repeat in the loop. Press Shift or Ctrl when clicking to select multiple items.
- ▶ Right-click the selected objects and select *Insert → Loop*.
- ▶ When prompted *Would you like to move selected objects into new loop?* (default setting), click *Yes*.
- ▶ The selected objects will now be indented in the Test Contents inside the loop.
- ▶ In the Test Element Details, enter the number of iterations that you want the loop to repeat.
- ▶ If you want the test elements to repeat at a specific rate, then select *Control the rate of iterations*.
 - Under *Options*, select the unit of time (millisecond, second, minute, or hour) from the drop-down list.
 - Enter a number for the iteration rate.
 - For a more realistic test, you can select *Randomly vary the delay between iterations*. With this selected the rate will vary from one iteration to the next but will average the specified rate over time.
 - For a more realistic test, you can also select *Delay before the first iteration of the loop* to stagger the first delay in each iteration (Figure 4-56).

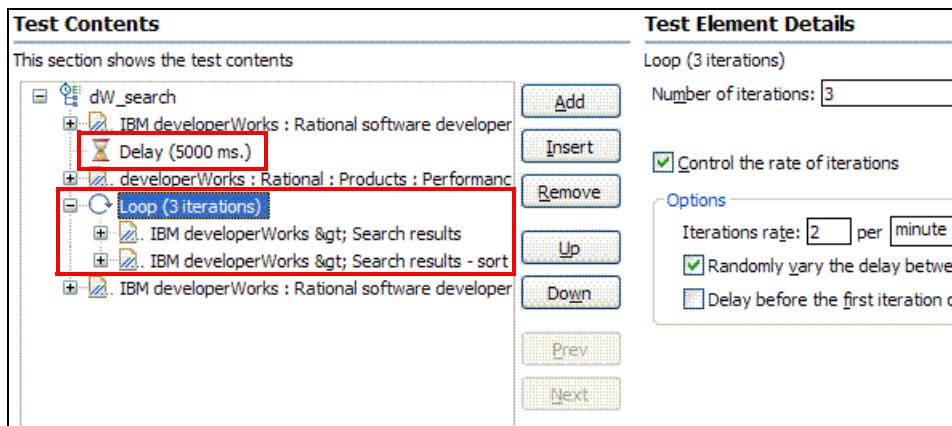


Figure 4-56 HTTP test with a delay and a loop

Loops in HTTP tests

One consideration when adding loops to HTTP tests is handling of cookies, and how the loops affect the state of virtual users. If verification points fail unexpectedly during a run, the cause might be that virtual users in loops did not maintain their original state. To enable each virtual user to enter the loop in the original state, you can modify the test's HTTP options or add custom code.

By default, the cookie cache for a virtual user is not reset during a test run. This is consistent with a browser's behavior. If a test or schedule contains loops, and a Web server sets a cookie during the first iteration of the loop, that cookie is *remembered* on subsequent iterations.

However, in certain instances, you might want to clear all cookies cached for a particular virtual user. For example, if you want each iteration of a loop to appear as a new user, you must reset the cache. If you do not, although the test completes, verification points that you have set within the test might fail.

There are two ways to reset the cookie cache, and each way has different effects.

To reset the cookie cache when looping in the schedule, or when the test follows another test in the schedule, use the following method. This resets the cache whenever the test is entered. Even if your tests do not loop, use this method if you are running back-to-back tests or Siebel tests.

- ▶ Open the test.
- ▶ In the HTTP options tab, select Clear cookie cache when the test starts.

To reset the cookie cache from one loop iteration to the next when you have put a loop around the entire contents of the test, and the loop is inside the test, add custom code to the test and call an API, as follows:

- ▶ Run the test or schedule to add the current Java libraries to the class path.
- ▶ Open the test and select the test element located at the point where you want the cookie cache to be reset. Typically, this is at the end of the loop.
- ▶ Click *Add* or *Insert* and select *Custom Code*. Add appends the custom code to the bottom of the selected element (test or test page). Insert adds the custom code above the selected page or page request.
- ▶ Add the following Java import statement:
`import com.ibm.rational.test.lt.execution.http.util.CookieCacheUtil;`
- ▶ Add the following Java code inside the exec method:
`CookieCacheUtil.clearCookieCache(tes);`

Example 4-1 shows a custom code addition that resets the cookie cache. The lines that you add to the generated custom code template are shown in bold.

Example 4-1 Example custom code to clear HTTP cookie cache

```
package test;

import com.ibm.rational.test.lt.execution.http.util.CookieCacheUtil;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

public class ClearCookies implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    public ClearCookies() {
    }
    public String exec(ITestExecutionServices tes, String[] args) {
        CookieCacheUtil.clearCookieCache(tes);
        return null;
    }
}
```

Loops in SAP tests

The only consideration when adding loops to SAP tests is that only transactions, screens or SAP events can be contained in a loop element. SAP test or session elements cannot be contained in a loop.

Loops in Citrix tests

Because of the nature of Citrix performance tests, it is important to preserve the sequence of events so that user actions remain in the correct context. Manual editing of Citrix performance tests is likely to break the context of user actions and produce synchronization time-outs and execution errors.

4.7.4 Conditions

Performance Tester allows you to add conditional logic to a test through a graphical condition element. This works the same as a simple if-then statement in software code. This allows a test to branch into two or more scenario paths based on events during execution, simulating a user choice. Like a loop element, this is added so that other elements, typically requests and responses, are put inside the condition (the *then* part).

A condition requires two string operands to compare and at least one is typically taken from the test itself. For example, you can check the value of a certain parameter by comparing it to a hard coded string (entered in the conditional's *Second operand*). Or you can compare two test parameters to one another and if they match, then the condition is met. To do this, you must use a reference that is made before (above in the test contents) the condition in the test. The test reference is similar to a variable used in the comparison. For information on creating a reference, see "Creating a reference" on page 83.

To add a  **Condition** to a test:

- ▶ Open the test.
- ▶ Select the test items (pages, screens, transactions, and so forth) that you want to execute if the condition is met. Press Shift or Ctrl when clicking to select multiple items.
- ▶ Right-click the selected objects and select *Insert → Condition (If)*.
- ▶ When prompted *Would you like to move selected objects into new IF?* (default setting), click *Yes*.

If you click *No*, then you will have to manually add pages, screens, windows, or other test elements into the conditional block.

- ▶ The selected objects will now be indented in the Test Contents inside the condition (Figure 4-57).

Figure 4-57 HTTP test with an If conditional block

Notice that you will not see an element labeled *Then* in the Test Contents, unless you add an Else block to the condition.

- ▶ To add an Else block:
 - a. Select the test items (pages, screens, transactions, and so forth) *inside the If block* that you want to execute if the condition is not met. Press Shift or Ctrl when clicking to select multiple items.

Tip: You might want to add test items to an Else block that are not included in the initial If block. In this case, you could delete the If element, clicking Yes when prompted *Would you like to keep child objects?*, then select all of the items you want to be part of the If and Else blocks and re-insert the conditional.

- a. Right-click and select *Insert → Condition (If) - ELSE Block*.
- b. When prompted *Would you like to move selected objects into the new ELSE block?*, click Yes.

If you click *No*, then you will have to manually add pages, screens, windows, or other test elements into the Else block (Figure 4-58).

The screenshot shows the 'Test Element Details' dialog for an 'If-Then-Else' conditional block. The 'Condition' section is set to 'If' with 'First operand: true', 'Operator: Equals', and 'Second operand: false'. The 'Options' section includes a checked 'Case-sensitive comparison' option. The 'Then' section lists 'IBM United States', and the 'Else' section lists 'IBM Deutschland'. A red box highlights the 'If' block in the tree view.

Figure 4-58 HTTP test with an If-Then-Else conditional block

- ▶ In the Test Element Details area, under Condition:
 - a. In the *First operand* field, either select the input for the block (a reference containing a string value to be compared with the *Second operand*, or a field reference to be used with the contains operator) or type a value.
The drop-down will list all existing references before (above) the conditional in the test. If you do not see the value you want to compare then you will have to create the reference and return to select the value.
 - b. In the *Operator* field, indicate the basis of comparison of the two operands (Table 4-2). Note that the two operands are strings.

Table 4-2 Conditional block operators

| Operators |
|------------------|
| Equals |
| Contains |
| Starts with |
| Ends with |
| Less than |
| Less or equal |
| Greater than |
| Greater or equal |

- To create a negation of the operators, for example, *Not equals* or *Does not contain*, check *Negate the operator* under *Options*.
- c. In the *Second operand* field, either select the input for the block (a reference containing a string value to be compared with the *First operand*) or type a value. When the defaults are used (both operand fields set to *true* and the *Operator* field set to *Equals*), the block is always processed.
 - ▶ In the Test Element Details area, under Options, choose the comparison type you want by selecting or clearing the check boxes.

Conditions in HTTP tests

There are no special considerations when adding conditional blocks to HTTP or Siebel tests that are unique to these types of tests.

Conditions in SAP tests

The only consideration when adding conditional blocks to SAP tests is that only transactions, screens or SAP events can be contained in a conditional block. SAP test or session elements cannot be contained in a conditional blocks.

Conditions in Citrix tests

Because of the nature of Citrix performance tests, it is important to preserve the sequence of events so that user actions remain in the correct context. Manual editing of Citrix performance tests is likely to break the context of user actions and produce synchronization time-outs and execution errors.

4.7.5 Transactions

A transaction element is used to combine other test elements together in a logical group. This allows you to get better data from logs and reports by measuring *user transactions*, in addition to the individual pages and requests. When viewing the test results, you can view performance data about any transactions that you have added. This is important because performance requirements are typically given in terms transaction, activities, or use cases from the user perspective.

For example, a system might provide a service *get address* that involves the user going through three screens/pages to complete. The performance testing effort is concerned first with the overall system response times for the *get address* transaction, which is the combined measurement of the 3 screens. The default Performance Tester reports would show times and statistics for each individual screen or page because it would not know what your performance requirements are. By grouping steps of a test together into transactions, the default reports would then show the measurements that you want.

To put a group of test elements into a  **Transaction**:

- ▶ Open the test.
- ▶ In the test, select the pages or page elements to group together. Use Shift+click to select multiple contiguous pages or page elements; use Control+click to select multiple non-contiguous items.
- ▶ Click *Add* (to place the transaction at the bottom of the test or page) or *Insert* (to place the transaction immediately before the selected item or block), and click *Transaction*.
- ▶ You are prompted whether to move the selected objects into the transaction. Click *Yes* or *No*.

In the transaction details, you can give the transaction a meaningful name. This is useful in the Transactions report, which lists transactions by name (Figure 4-59).



Figure 4-59 HTTP test with several transactions

Creating a custom transaction

Another possible use of the transaction element is to define customized transactions that cannot be easily recorded. This is not common but it does allow you to overcome potential performance testing obstacles. You would typically use custom code to emulate response times or other transaction delays, but you could also manually add pages and requests to complete a custom transaction.

The general steps to adding a custom transaction are as follows:

- ▶ Open the test you want to add the transaction to.
- ▶ Select the element in the test just *after* where you want to add the transaction.

- ▶ Right-click and select *Insert → Transaction*.
- ▶ When prompted *Would you like to move selected objects into new loop?* (default setting), click *No*. You will now have an empty transaction.
- ▶ Right-click on the transaction and select *Add → Custom Code*.
Alternately, you could add a page or other element within the transaction.
- ▶ Edit the code or other elements to emulate the required transaction or timing.

Figure 4-60 shows a custom transaction implemented with custom code.

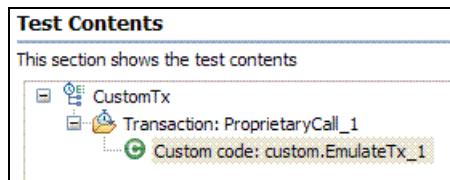


Figure 4-60 A custom transaction

4.7.6 Custom code

You can add any number of custom Java code classes to a test to provide additional functionality or extend the test's capabilities in a number of different ways. Chapter 13, “Testing with custom Java code” on page 4291 explains Performance Tester's custom code in more detail; this chapter only briefly discusses how they are added to tests.

Custom code is typically created from within a test, as if it were part of the test. However, custom code is saved as Java files separate from the test, and different tests can include calls to the same custom code. Editing custom code is the only time when editing a Performance Tester test that you will be working directly with software code.

Placement and references

A custom code module is executed similar to a function call in procedural software code. Custom code modules are executed at the point that they are inserted into the test, so therefore the placement within the test is very important.

You have to consider what data, if any, you have to pass to the code as arguments. To pass a value from the test to custom code, you must first create a reference for that value (see “Creating a reference” on page 83) and that reference must occur before (above) the custom code in the test.

Creating and adding new custom code

When you know the proper location, you can add or insert custom code just as you would any other test element. Adding the custom code will create the new Java class and allow you to edit the code within.

To add new  **Custom Code** to a test:

- ▶ Open the test.
- ▶ Select the element in the test just *after* where you want to place the custom code.
- ▶ Right-click and select *Insert → Custom Code*.
- ▶ In the Test Elements Details, enter a *Class name*.

Performance Tester will automatically generate a class name in the test package (prefixed before the name) but you should always give your custom code a meaningful name. We also recommend that you change the default *test* prefix in the name to a package name that you create.

Tip: You can better organize custom code by storing it in its own package (folder). The simplest way to do this is to prefix the *Class name* with a package name of your choosing, such as “customcode.”, with a period separating the package name from the class name. You must use a period between the two names or the custom code file will be placed in the project root. For example, entering a custom code *Class name* of **“customcode.MyCode”** will automatically create a package (folder) named **customcode** plus a Java file named **MyCode** inside this package.

When you create a custom code package, you can use the same name for all subsequent custom code that you create. If needed, you could also create multiple packages to store different custom code files.

Doing this will keep the custom code Java files that you edit separate from the Performance Tester generated Java files which you do not edit. Java naming conventions should be followed for packages and class names.

- ▶ Under the *Arguments* field, click *Add*.
- ▶ In the *Custom Code - Select Arguments* dialog (Figure 4-61), select the values you need to pass as arguments.

The list will show existing references before (above) the conditional in the test, attached datapool values, and any custom code return values. If you do not see the value you want to compare then you will have to create the reference and return select the value.

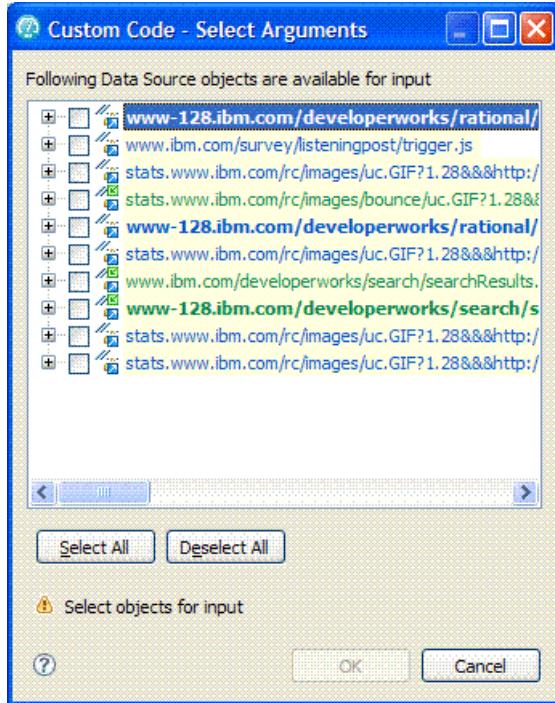


Figure 4-61 Selecting arguments for custom code

- ▶ Click *Generate Code* to create the custom code and open the Java editor (Figure 4-62).

At this point you are ready to edit the code itself; refer to Chapter 13, “Testing with custom Java code” on page 429 for information on the Performance Tester custom code interface.

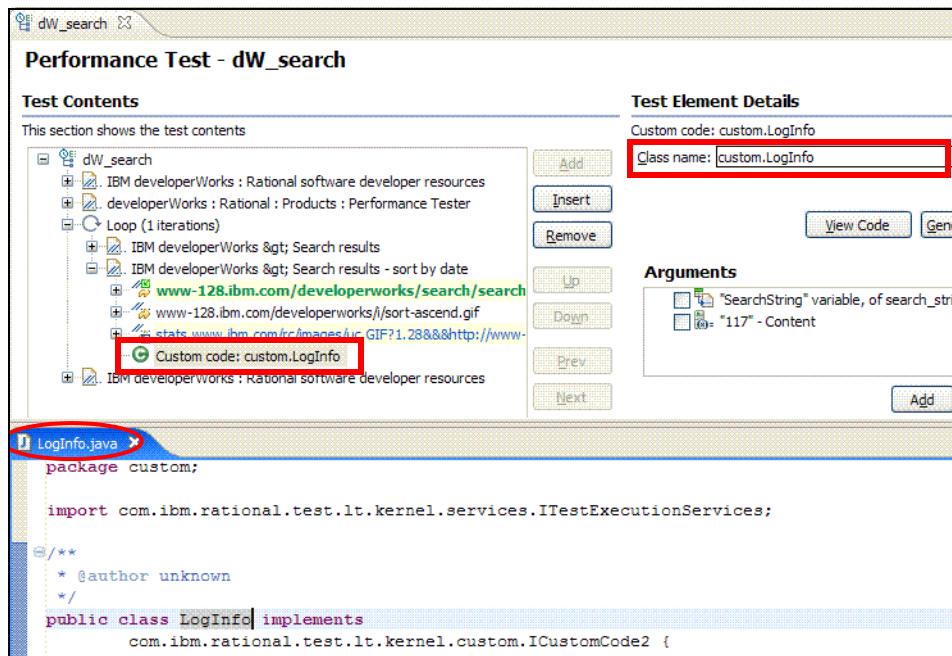


Figure 4-62 Adding custom code to a test

Reusing existing custom code

In some cases you might have to add custom code to do something that you have already implemented in other tests. In this case you can simply reuse existing custom code.

To reuse existing  **Custom Code** in a test:

- ▶ Open the test.
- ▶ Select the element in the test just *after* where you want to place the custom code.
- ▶ Right-click and select *Insert* → *Custom Code*.
- ▶ In the Test Elements Details, in the *Class name* field, enter the exact same package and name of the existing custom code.
- ▶ Click *View Code* to open the Java editor and verify that the class contains the code you intend to use.

If you click *View Code* and see an empty class (you will always see the basic class structure and declarations), then you did not enter the correct name or package of the existing custom code. Verify the package location and class name, then try again.

4.8 Adjusting connections and timing

A Performance Tester test will emulate many server connections along with time delays and transaction rates. These connections and timings are typically captured from the recorded session. Similar to the test data captured from a recorded session, you might need to modify these connections or timing to create more varied and realistic tests.

4.8.1 Modifying timing

You will typically manipulate timing in a schedule to emulate specific workload rates and pacing (see Chapter 6, “Scheduling tests” on page 155). There are circumstances where you might modify timing in tests, however, the settings in a schedule will usually override the timing in a test. For example, if a test contains a particular think time delay of 5,000 milliseconds and the schedule containing the test has a maximum think time of 3,000 milliseconds, then executing the schedule will limit the test’s think time to 3,000 milliseconds, disregarding the value saved in the test.

Think times

Think times emulate user pauses between system activity. This is usually the time a user spends reading or thinking between seeing a response and clicking or triggering the next request.

Because think times represent a user’s time and not the system’s time, they will not directly affect measured system performance. A test with very long think times might still have very small response times, and the resulting performance reports will only reflect the system’s response times. Think times can, however, impact response times by affecting the rate and load that a system experiences during test execution. Tests that have a lot of very long think times will give a system more time to *recover* and respond to subsequent requests faster than a test with smaller think times that simulate users hitting a system with rapid successive requests.

Think times versus delays

In addition to think times, you can also add pauses using the delay test element. Delays and think times are similar in that both create pauses between requests and responses, and they are not counted as time waiting for a system response. However, they are different in several ways and you should use the correct one accordingly. You should use think times when you want to emulate the delay caused by a user. Refer to “Delays versus connection think times” on page 107 for a description of the differences between delays and think times.

For HTTP and Siebel tests, think times are captured at the page element level (Figure 4-63). For SAP tests, think times are captured at the transaction element level (Figure 4-64). Citrix tests handle think times differently to ensure proper synchronization, and you do not change these values directly.

The screenshot shows the 'Test Element Details' dialog box. At the top, it displays the path: developerWorks : Rational : Products : Performance Tester. Below that, the 'Page title' is set to 'developerWorks : Rational : Products : Performance Tester'. The 'Primary request' is listed as 'www-128.ibm.com/developerworks/rational/products/performancetester/'. Under the 'Think time' section, there is a text input field containing '8312' and a dropdown menu set to 'milliseconds'. A red box highlights this entire section.

Figure 4-63 Think times in an HTTP test

To edit think times in a test:

- ▶ Open the test.
- ▶ In the test editor, select the test element (HTTP page, SAP transaction) with the think time that you want to edit.

Think times are associated with the element that occurs *after* the user pause. For example: if a user starts with *Page A*, waits 4 seconds, then goes to *Page B*, the generated test would show a think time of 4,000 milliseconds in the Test Element Details of *Page B*.

- ▶ Under the Test Element Details, modify the think time value as needed.

SAP think times

Think times can be edited for any get or set event and are basically the same as think times in other types of tests.

The screenshot shows the 'Test Element Details' dialog box for an SAP test. It includes a 'SAP element label' section with a button to 'Set "GuiOkCodeField" text to "fs10n"' and a 'Restore Default' button. Below this is a 'Think time' section with a text input field containing '1446' and a dropdown menu set to 'milliseconds'. A red box highlights this section. Further down, there is a 'SAP Set' section with fields for 'Property name' (set to 'text') and 'Value' (set to 'fs10n').

Figure 4-64 Think times in an SAP test

Response delays

Response delays, which are different from the *Delay* test element (see 4.7.2, “Delays” on page 107), emulate system and internal delays during system activity. An example of this is the amount of time that a client application takes to process the received protocol data and render it to the user. These times are almost always very small, especially compared to think times.

Because these delays represent the system’s time, they do factor in to the measured system performance. The delay values are captured from the recorded sessions and, except for rare cases, you will not edit these values.

HTTP delays

Delays in HTTP tests are saved as attributes of a request (Figure 4-65). These are typically very small and occur *before* the requests is made. The majority of HTTP requests in a test will not have any delay, but you might find them after the primary page response, before a request for a JavaScript or cascading style sheet (css) file for the page.

Test Element Details

www-128.ibm.com/js/dropdown.js

Request Attributes

| | | | |
|----------|-----------------|---------|-----|
| Version: | 1.1 | Method: | GET |
| Host: | www-128.ibm.com | Port: | 80 |
| URL: | /js/dropdown.js | | |

Data:

Request Headers

| Header Name | Value |
|-------------|---|
| Host | www-128.ibm.com |
| User-Agent | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-... |

- Click to set as primary Delay: milliseconds

Character set: utf-8  UTF-8: Unicode UTF-8

Connection www-128.ibm.com:80 Response 200 - OK

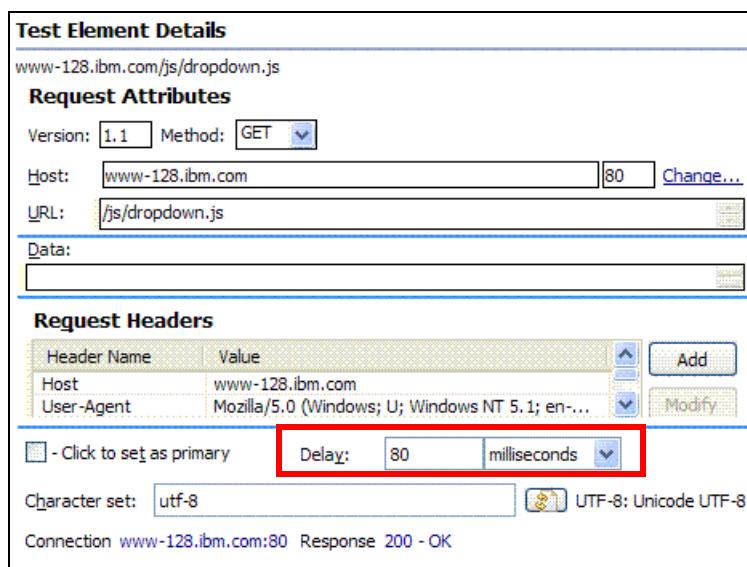


Figure 4-65 HTTP delay

SAP delays

Delays in SAP tests are saved as attributes of a server request and are called *Interpretation times* (Figure 4-70 on page 127). This is the delay between the moment the data is received by the SAP GUI client and the moment the screen is updated. It measures interpretation of data by the SAP GUI client, and not SAP R/3 server performance. These are typically very small and there is usually some interpretation time for each server request.

Timing in Citrix tests

Citrix tests contain the same kind of think times or delays as HTTP/Siebel or SAP tests. A test must be carefully synchronized for test execution, and the only thing you would typically do in a Citrix test to affect this is to define or edit the *Response Time* measurements. Editing response time definitions are briefly discussed in “Measured response times” on page 102. For information on Citrix test timing and synchronization, see Chapter 12, “Testing Citrix application environments” on page 401.

Emulating transaction rates

While performance testing, usually you need to create tests that will simulate workload scenarios with a specific transaction rate, for example: *50 users perform this transaction every hour*. Typically you would configure these transaction rates in schedules, not tests (see 4.1.2, “Editing and augmenting tests versus schedules” on page 42 and Chapter 6, “Scheduling tests” on page 155). There might be cases, however, where you can set a transaction rate in a test by configuring a loop element iteration rate (see 4.7.3, “Loops” on page 108).

4.8.2 Modifying connections

There are many parameters that affect connections between clients and servers. Some of these might be relevant to performance testing and could impact system performance. You can modify these parameters in Performance Tester to better model users, simulated various networking scenarios, or satisfy performance testing requirements.

Editing HTTP connections

You can edit many different parameters in Performance Tester to modify HTTP connections.

Server connections

Unlike SAP or Citrix, a Web-based application can make many connections to different servers within a single test, even within a single page. Each connection is represented by an icon in the Test Contents under the request that it is associated with. By selecting the connection, you can see all of the requests in the test that use the connection (Figure 4-66).

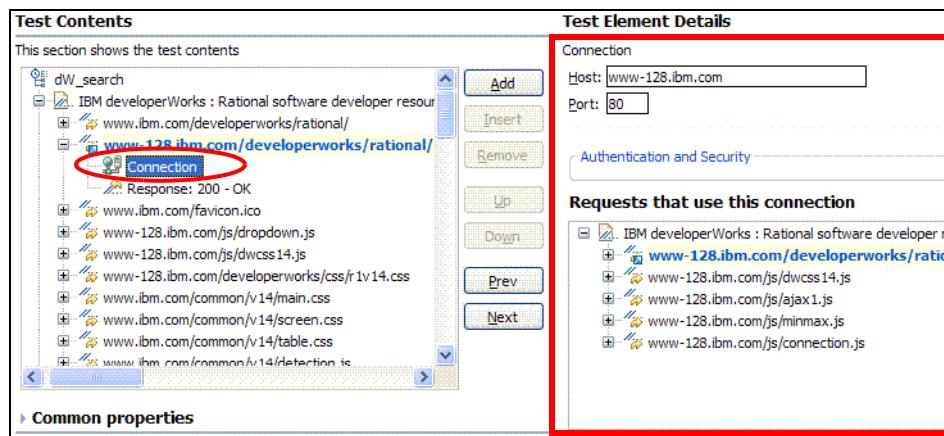


Figure 4-66 HTTP server connection

Headers and requests

You can modify the contents of the headers that are contained in HTTP requests and responses for HTTP or Siebel tests. This allows you to change many different parameters pertaining to the communication and connections between client and server. This is typically done with requests but could also be done with responses.

To change the values in an HTTP header:

- ▶ Open the test.
- ▶ In the Test Contents, click a request or response.
- ▶ In the Test Element Details area (Figure 4-67), locate the *Request Headers* table (or *Response Headers* if you selected a response), and double-click the cell containing a value that you want to edit.

The table view will change to an edit field where you can modify the header value (Figure 4-68):

- After you have finished editing the value, you can click *Table View* to return to the initial table showing all header values.
- You can click *Prev* or *Next* to go to the previous or next header values.

Test Element Details
 www-128.ibm.com/developerworks/rational/

Request Attributes

| | | | |
|----------|--|---------|-----|
| Version: | 1.1 | Method: | GET |
| Host: | www-128.ibm.com | Port: | 80 |
| URL: | /developerworks/rational/ | | |
| Data: | <input type="text"/> <small>HTTP SOAP WS-Security</small> | | |

Request Headers

| Header Name | Value |
|-----------------|--|
| Host | www-128.ibm.com |
| User-Agent | Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; ...) |
| Accept | text/xml,application/xml,application/xhtml+xml,te... |
| Accept-Language | en-us,en;q=0.5 |
| Accept-Encoding | gzip,deflate |
| Accept-Charset | ISO-8859-1,utf-8;q=0.7,*;q=0.7 |
| Keep-Alive | 300 |
| Connection | keep-alive |

- Primary request for page Delay: 0 milliseconds

Character set: UTF-8  UTF-8: Unicode UTF-8

Connection www-128.ibm.com:80 Response 200 - OK

Figure 4-67 Header values for an HTTP request

Test Element Details
 www-128.ibm.com/developerworks/rational/

Request Attributes

| | | | |
|----------|--|---------|-----|
| Version: | 1.1 | Method: | GET |
| Host: | www-128.ibm.com | Port: | 80 |
| URL: | /developerworks/rational/ | | |
| Data: | <input type="text"/> <small>HTTP SOAP WS-Security</small> | | |

Header 4 of 8: Accept-Language

Value: en-us,en;q=0.5

Figure 4-68 Individual header value for an HTTP request

- ▶ Save the test.

To add a new value to an HTTP header:

- ▶ Open the test.
- ▶ In the test hierarchy, click a request or response.
- ▶ Under the Test Element Details area, locate the *Request Headers* table (or *Response Headers* if you selected a response), then click *Add*. The Add/Edit Headers window opens (Figure 4-69).

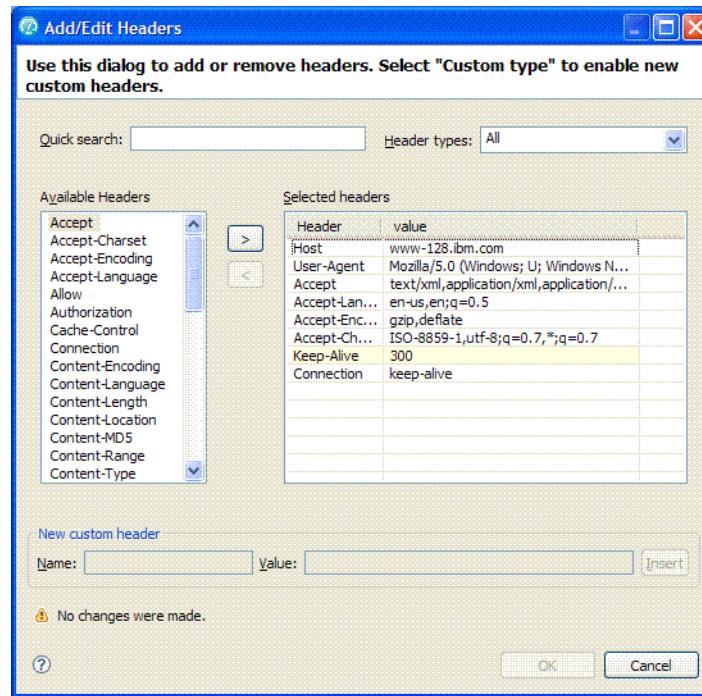


Figure 4-69 Adding or editing HTTP header values

- ▶ To add a standard header:
 - In the *Available Headers* list, locate the header to add and click it. Use the *Quick search* field (start typing the name of a header) and *Header types* list (select the type of header you are looking for) to quickly locate a header in the *Available Headers* list.
 - Click the right angle bracket (>). The selected header moves into the *Selected headers* list and your cursor is placed in the value column.
 - Type the value for the header.

- ▶ To add a custom header:
 - In the *Header types* list, select *Custom*.
 - At the bottom of the window, in the *New custom header* area, type the header information in the *Name* field and the *Value* field, and then click *Insert*. The custom header is added to the *Selected headers* list.
- ▶ When you have finished adding headers, click *OK*.

Editing SAP connections

Compared to HTTP tests, there are fewer connection parameters that you can edit in SAP tests. You are able to edit some connection elements for the entire test, server request details, and think times for SAP get and set events.

SAP connection details

In the test editor, SAP connection elements are at the top of the test site and describe the connection to the SAP R/3 server. These settings apply to the entire test.

SAP server request details

In the test editor, server request elements are located at the end of every SAP screen and provide information returned by the server for the selected screen. All fields in a server request element are read-only (Figure 4-70).

| Name: | FS10N |
|---------------------------|-------------|
| Program: | RFGLBALANCE |
| Flushes: | 1 |
| Response time (ms): | 625 |
| Interpretation time (ms): | 110 |
| Roundtrips: | 2 |

Figure 4-70 SAP server request details

4.8.3 Changing environments for testing

It is likely that the performance tests you initially develop might need to be executed later in a different system environment with different hosts. For example, you might record tests in the test environment but later run them against the production environment. You might also have different host machines to test within a given environment, or different configurations such as varying ports which have to be tested. While this host and configuration information is saved as part of a test, you can still reuse and share tests throughout the entire testing effort.

Changing hosts in HTTP tests

To reuse HTTP tests on different hosts, you must change the following fields:

- ▶ The host name of the server connection in requests
- ▶ The port number of the server connection in requests
- ▶ The host name and port number in the host header field in requests
- ▶ The URL of the redirected requests, if you are using a proxy to record

You can change these by using the search and replace functions. The following steps show how to change the pertinent test elements.

- ▶ To change the host on all requests:
 - Open the test, right-click, and select *Test Search*. The Search window opens.
 - In *Search for text*, type the value of the original host.
 - In *Elements to search*, select and highlight *HTTP Requests*.
 - In *Fields*, select *Host* and click *Replace*. The Replace window opens.
 - In the *With* field, type the value of the new host, and click *Replace All*. You have now changed the hosts on all of the requests.
- ▶ If the new host has a different port, change the port number as follows:
 - Open the test, right-click, and select *Test Search*. The Search window opens.
 - In *Search for text*, type the original port, which is usually 80.
 - In *Elements to search*, select and highlight *HTTP Requests*.
 - In *Fields*, select *Port*, and click *Replace*. The Replace window opens.
 - In the *With* field, type the new port, and click *Replace All*. You have now changed the numbers of the ports in all of the requests.

- ▶ To change the host name and port number in the host header fields:
 - Open the test, right-click, and select *Test Search*. The Search window opens.
 - In *Search for text*, type the original header.

Note: If you recorded using a nondefault port, type both the host and port, separated by a colon (host: port). If you recorded using default port 80, or your host name does not contain a port, type only the recorded host name.
- In *Elements to search*, select and highlight *HTTP Requests*.
- In *Fields*, select *Headers Values*, and click *Replace*. The Replace window opens.
- In the *With* field, type the new host: port (if you recorded using default port 80, type the host name only), and click *Replace All*. You have now changed the headers in the test.
- ▶ If you recorded the test using a proxy, you must change the URLs as well. To change the URLs:
 - Open the test, right-click, and select *Test Search*. The Search window opens.
 - In *Search for text*, type the original URL.
 - In *Elements to search*, select and highlight *HTTP Requests*.
 - In *Fields*, select *URL*, and click *Replace*. The Replace window opens.
 - In the *With* field, type the new URL, and click *Replace All*. You have now changed the URLs in the test.

After making any of these edits to a test, rerun the test and inspect your results to confirm that the test ran using the expected host and port.

Changing hosts in SAP tests

There are two ways that a Performance Tester test can connect to an SAP system: using the system name or using a connection string. Both are captured into a test when you record an SAP session, however, you can select which method Performance Tester will use to execute the test.

To change the server for an SAP test:

- ▶ Open the test.
- ▶ In the Test Contents, select the SAP connection element. This is the first element after the test name.

- ▶ Under Test Elements Details, make one of the following changes:
 - Enter the new system name in *SAP system name* and clear *Connection by string*.
 - Enter the new connection string and select *Connection by string*.

You could also change both the system name and the connection string, although only one will be used for test execution, depending on whether *Connection by string* is checked or not.

If you change the SAP system name, you will have to configure the connections on every test agent machine used for test execution. The system name is used by the SAP Logon application to start the SAP GUI.

If you change the connection string, you only have to change the test and not the connections for every test execution machine. However, it can be complicated to know the correct router string, and we recommend that an SAP administrator help provide correct connection strings.

Changing hosts in Citrix tests

To reuse Citrix tests on different servers, you must first be sure that screen resolution, colors, task bar look and feel, and other graphics settings are identical on all the servers on which you want to run the tests. If there are any differences in appearance or timing, then it will not be feasible to reuse the test.

To change the server for a Citrix test:

- ▶ Open the test.
- ▶ In the Test Contents, select the *Citrix session*. This is the first element after the test name.
- ▶ Under Test Elements Details, for Server Address, enter the new server. This value is marked as a datapool candidate by default.

4.8.4 Manipulating authentication

Along with changing the environment of your performance tests, you might also need to change the ways that test users are authenticated to the system under test. It is common to use test user accounts for performance testing, and these might change in time, or as the hosts of environments change for test execution. While there are a number of ways to manipulate authentication in tests, Performance Tester provides specific features for this with HTTP tests.

Adding authentication in HTTP tests

Web application servers can include an option to force a login. You might have recorded a test with this option disabled but want to run the test with the option enabled. Adding an authentication folder to the appropriate test request lets you do this without recording the test again.

To add an authentication folder  to an HTTP request:

- ▶ Open the test.
- ▶ Click the request that will pass the authentication parameters.
- ▶ Click *Add* and select *Basic Authentication*. A folder named Authentication is added to the request, and the Test Details area displays the Userid, Password, and Realm fields (Figure 4-71).

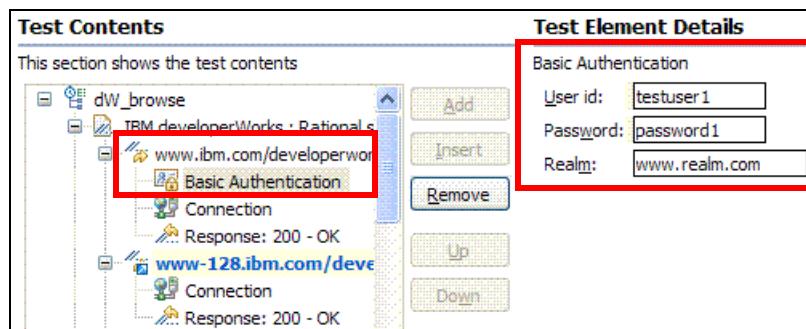


Figure 4-71 Basic authentication for an HTTP request

HTTP Digital certificate overview

If you want to use digital certificates to run HTTP tests against applications that require client-side digital certificates to authenticate users, work with the appropriate server administrators to determine the types of certificates that you have to create. There are four types of certificates that can be used in performance testing.

- ▶ **Self-signed certificates.** These are used when no entity needs to vouch for the authenticity of the certificate. These are the simplest certificates to create and use. Typically, however, a signed certificate is used to represent a particular user.
- ▶ **Signed certificates.** These are used when a certificate needs to be created for and issued to one, and only one, user. Signed certificates are signed by a certificate authority (CA).
- ▶ **Certificate authority (CA) certificates.** These are self-signed certificates used to sign (certify) certificates.

- ▶ **Unsigned certificates** (rarely used). These are certificates that are neither signed by a CA nor self-signed. Most Web applications do not use unsigned certificates.

When you create a self-signed or signed certificate (including CA certificates), you can specify a *subject*. The subject of a certificate is the set of attributes of an X.500 Distinguished Name that is encoded in the certificate. The subject allows the recipient of a certificate to see information about the owner of the certificate. The subject describes the certificate owner, but is not necessarily unique. Think of subjects as entries in a telephone book; there can be multiple entries for John Smith, but each entry refers to a different person.

The subject can contain many different types of identifying data. Typically, the subject includes the following attributes (Table 4-3).

Table 4-3 Digital certificate subject attributes

| Attribute | Example |
|-------------------------------|--------------------------------|
| COMMON NAME (CN) | CN=John Smith |
| ORGANIZATION (O) | O=IBM Corporation |
| ORGANIZATIONAL UNIT (OU) | OU=IBM Software Group |
| COUNTRY (C) | C=US |
| LOCALITY (L) | L=Chicago |
| STATE or PROVINCE (ST) | ST=IL |
| E-MAIL ADDRESS (emailAddress) | emailAddress=smith@abc.ibm.com |

This information can be typed as one string, using forward slashes to separate the data. In our example, the subject would be typed as follows:

```
/CN=John Smith/O=IBM Corporation/OU=IBM Software Group  
/C=US/L=Chicago/ST=IL/emailAddress=smith@abc.ibm.com
```

Creating a digital certificate store

You can use the KeyTool program supplied with Performance Tester to generate a list of names of certificates in a certificate store. The KeyTool command-line program enables you to create a Rational Certificate Store (RCS) file that contains digital certificates for use with performance tests. A Rational Certificate Store (RCS) file is a compressed archive file that contains one or more PKCS#12 certificates. You can also use the KeyTool program to remove certificates from a certificate store.

To create a certificate store:

- ▶ From the command line, type the following command:

```
java -cp rpt_home/plugins/com.ibm.rational.test.lt.kernel_version.jar  
      com.ibm.rational.test.lt.kernel.dc.KeyTool --store=file  
      --passphrase=certificate-passphrase --add --remove --generate  
      --cert=certificate-name --subject=subject-name --ca-store=store  
      --ca-cert=ca-certificate-name --ca-passphrase=ca-certificate-passphrase  
      --sign --self-sign --algorithm=algorithm {RSA | DSA} --list
```

If a value contains spaces, enclose the value in quotation marks. Each option is described in Table 4-4.

Table 4-4 KeyTool command-line options

| Option | Description |
|-----------------|---|
| --store | Required if adding or removing a certificate. The file name of the Rational Certificate Store (RCS) file. If the specified certificate store does not have the RCS extension, this extension will be added. |
| --passphrase | Optional. The passphrase to place on the generated certificate. The default passphrase, which is also the passphrase required during playback, is default. |
| --add | Optional. Adds the certificate to the certificate store. Used with --generate, this generates a certificate and adds it to the certificate store. |
| --remove | Optional. Removes the certificate from the certificate store. This option cannot be used with the --add or --generate options. |
| --generate | Optional. Generates a certificate. Used with --add, this generates a certificate and adds it to the certificate store. |
| --cert | Required. The name of the certificate file to add, remove, or generate. If you are creating a certificate, the file name will be given the P12 extension. |
| --subject | Optional. The X.500 Distinguished Name for the certificate. If no subject is specified, a default subject will be provided. To learn more about subjects, see Digital certificate creation overview. |
| --ca-store | Required if signing a certificate. The file name of the Rational Certificate Store (RCS) file from which to retrieve the CA certificate. |
| --ca-cert | Required if signing a certificate. The name of the CA certificate file to use to sign another certificate. |
| --ca-passphrase | Required if signing a certificate. The passphrase for the CA certificate. |

| Option | Description |
|-------------|--|
| --sign | Optional. Signs the generated certificate using the specified CA certificate. This option cannot be used with --self-sign. |
| --self-sign | Optional. Self-sign the generated certificate. This option cannot be used with --sign. |
| --algorithm | Optional. This determines the encryption algorithm to use. The default is RSA. The options are RSA or DSA. |
| --list | Optional. This prints the names of all certificates in a certificate store to standard output. This list can be used to create a datapool. |

- ▶ Use the KeyTool to create and add as many digital certificates as you want. If you want to create a datapool of the names of certificates in the certificate store, run KeyTool again with the --list option. This writes a list of names that can then be imported to a datapool.

You now have a digital certificate store that you can use with performance tests. Because the KeyTool program has many options, you might want to create an alias or script file to use to invoke the KeyTool.

You do not have to use the KeyTool command-line program to create a certificate store. It is possible to use existing PKCS#12 certificates with Performance Tester. PKCS#12 certificates can be exported from a Web browser. PKCS#12 certificates encode the private key within the certificate by means of a password.

Important: Because the password is required to be default when you play back a recording in Rational Performance Tester, you must not use certificates associated with real users. Certificates associated with real users contain private keys that should not become known by or available to anyone other than the owner of the certificate. An intruder who gained access to the certificate store would have access to the private keys of all certificates in the store. For this reason, you must create, or have created for you, certificates that are signed by the correct certificate authority (CA) but that are not associated with real users.

Adding authentication in SAP tests

Authentication to an SAP system is captured in a Performance Tester test simply as setting text in an SAP element using an SAP Set. The values for a user name and password are identified as datapool candidates and can be substituted by datapools, custom code, or other available data sources. There are no other specific features for manipulating authentication in SAP tests.

Adding authentication in Citrix tests

Authentication to a system in Citrix is captured in a Performance Tester test simply as a Citrix text input. The values for a user name and password can be substituted by datapools, custom code, or other available data sources. There are no other specific features for manipulating authentication in Citrix tests.

4.9 Improving test reuse and documentation

So far, we have discussed various ways of editing tests to modify, complete, enhance or extend their capabilities. Another purpose of editing tests is to improve their readability and potential reuse. This is done primarily by adding comments and descriptions and possibly by renaming elements for better understanding.

4.9.1 Comments and descriptions

Although Performance Tester tests are graphical and therefore less cryptic than software code, they can still benefit tremendously from added comments. There are a number of places where comments and descriptions can clarify the purpose and function of a test's components making it much easier for someone else, other than the test's developer, to understand. This can greatly increase the value of the test as a software development artifact.

Adding comments

A comment in a Performance Tester test is another test element like those described in 4.7, “Adding test elements” on page 105. This is the same as a comment in software code. You can add as many comments as you like to a test without affecting the execution.

To add a  **Comment** to a test:

- ▶ Open the test.
- ▶ Select the location where you want to place the comment.
- ▶ Right-click and select *Insert → Comment*.
- ▶ In the Test Elements Details, enter comments into *Comment text*.

You can enter and view very long comments from the Test Elements Details, however, only the first 48 characters will be visible in the Test Contents.

- ▶ Save the test. Figure 4-72 shows a test with a comment and a description.

Tip: Another crucial place to add comments is within custom code modules, which are simply Java comments.

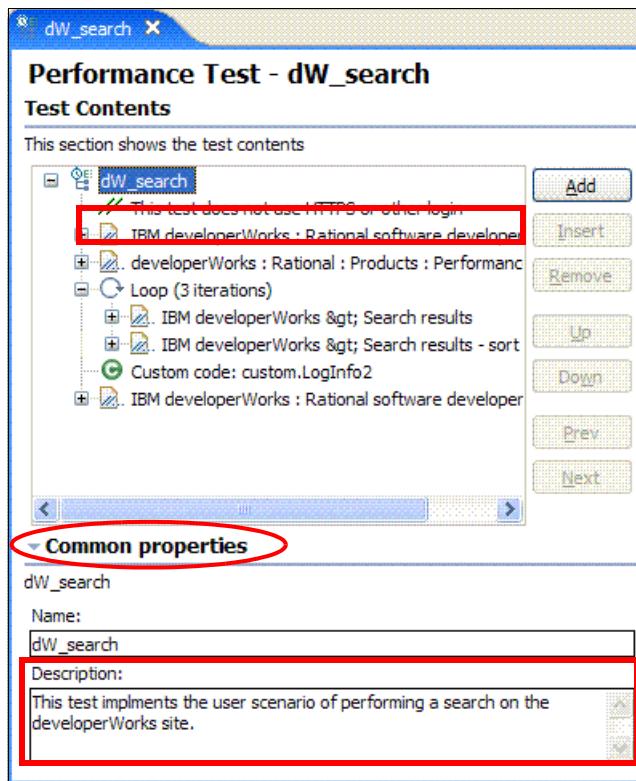


Figure 4-72 A test with comments and description

Adding descriptions

In a given testing effort, you create many tests and datapools, and there will most likely be other people who will use these artifacts. Descriptions should be added to every test and datapool to explain its purpose, usage, and any other relevant information.

The best time to add a description is when you first create the artifact. If you have to revise or extend a description, you can do so as described here (Figure 4-72):

- ▶ Open the test.
- ▶ Expand the *Common properties* area in the lower-left corner of the test editor by clicking on the expand arrow.
- ▶ In the expanded Common properties, enter text in the *Description* field.

To add a description to a datapool:

- ▶ Open the datapool.
- ▶ Display the *General Information* area by clicking on the *Overview* tab in the lower-left corner of the datapool editor.
- ▶ In the General Information area, enter text in the *Description* field (Figure 4-73).

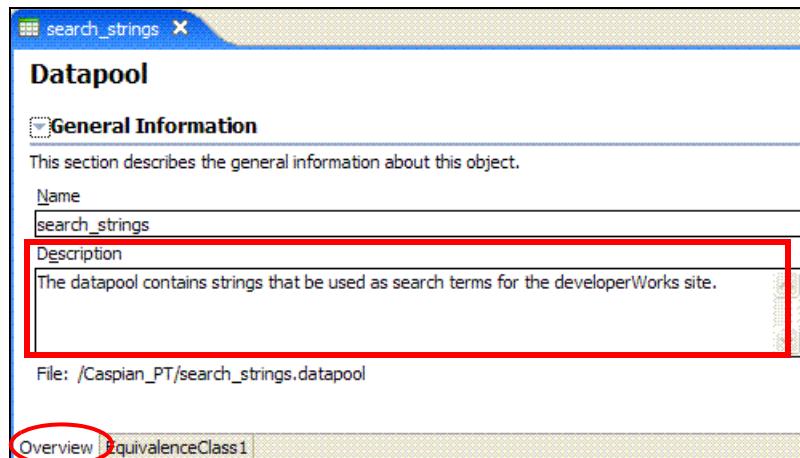


Figure 4-73 A datapool description

4.9.2 Improving test readability

In addition to adding comments and descriptions, you can improve the readability of a test by renaming elements to more accurately reflect their meaning or purpose. This is perhaps most commonly done to HTTP page titles but can also apply to transactions and custom code modules.

To rename an HTTP page title:

- ▶ Open the test.
- ▶ Select the page you want to rename in the Test Contents area.
- ▶ Click to place the cursor in the *Page title* field in the Test Elements Details area.
- ▶ Rename the page title and save the test.



Test playback and debugging

This chapter provides information about initiating test playback, monitoring test playback, and debugging.

We cover the following topics:

- ▶ Test playback
- ▶ Schedule playback
- ▶ Playback status
- ▶ Monitor and control
- ▶ Debugging
- ▶ Command line interface

5.1 Test playback

Producing realistic and sophisticated load tests requires using a schedule to model the workload you are trying to simulate and to specify the dedicated computers that will carry out the work.

Although in the long term you will want to use schedules, it is possible to playback a test without creating a schedule. To quickly verify test integrity, you can execute a test immediately after creating it.

To execute a test without a schedule, right-click on the test, then select *Run As* → *Performance Test*. RPT will then launch a one-user playback of the test.

5.2 Schedule playback

An RPT schedule provides the capability to specify the workload you want to simulate. This chapter introduces schedules as a primary way to launch and control test playback. Chapter 6, “Scheduling tests” on page 155 describes schedules in detail.

5.2.1 Create schedule

To create a schedule, select *File* → *New* → *Performance Schedule*. The Schedule File Name and Location window appears. Give your schedule a name. For example, enter SimpleSchedule, select the project name, and click *Finish* (Figure 5-1).

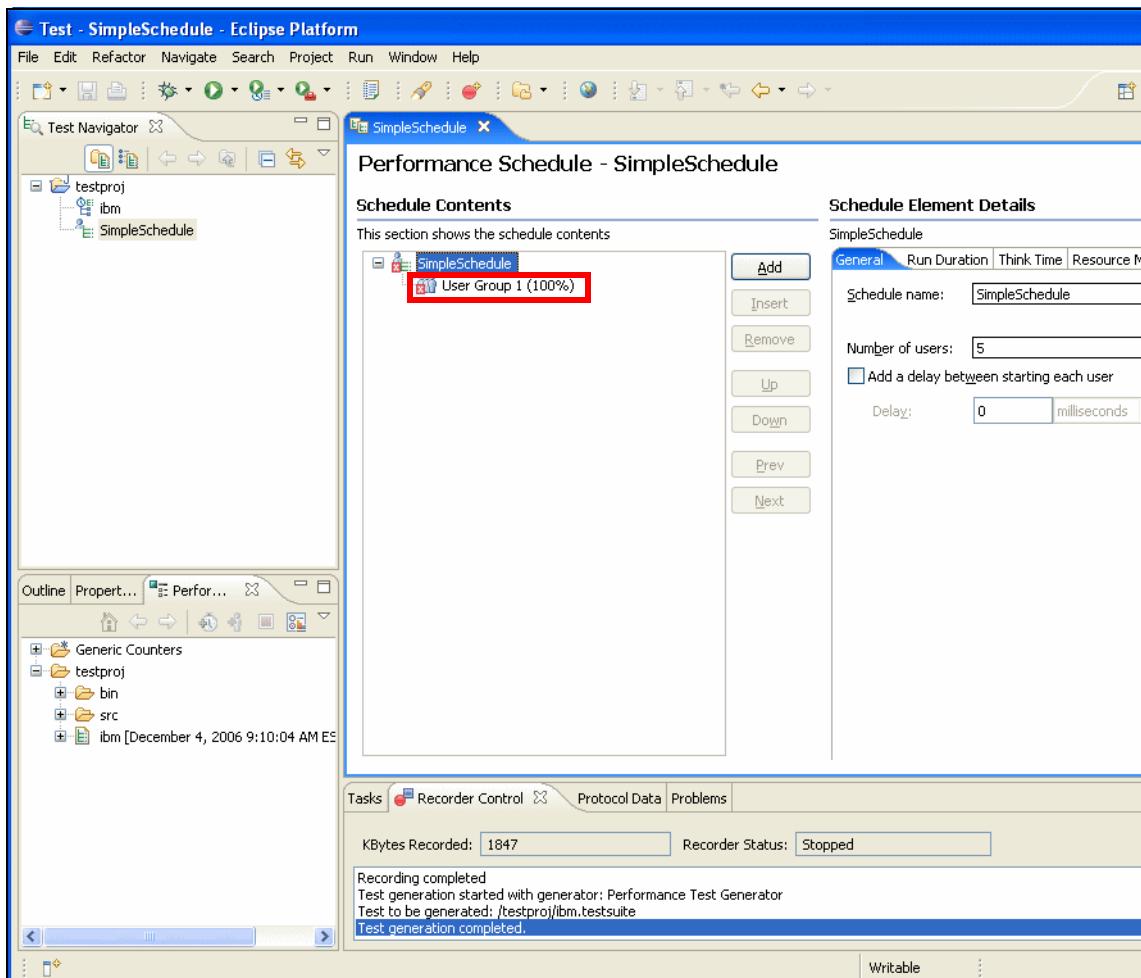


Figure 5-1 SimpleSchedule performance test schedule

A schedule requires, at minimum, one test to execute. Select *User Group 1* and click *Add*, then click *Test*. When the *Select Performance Tests* dialog appears, select a test and click *OK*. The test should appear under *User Group 1*. Save the schedule by selecting *File* → *Save*.

5.2.2 Playback schedule

To playback SimpleSchedule with your test, right-click *SimpleSchedule* in the Test Navigator window and select *Run As → Performance Schedule*. RPT then launches and executes SimpleSchedule. You can view the state of schedule playback in the Overall Status window (Figure 5-2).

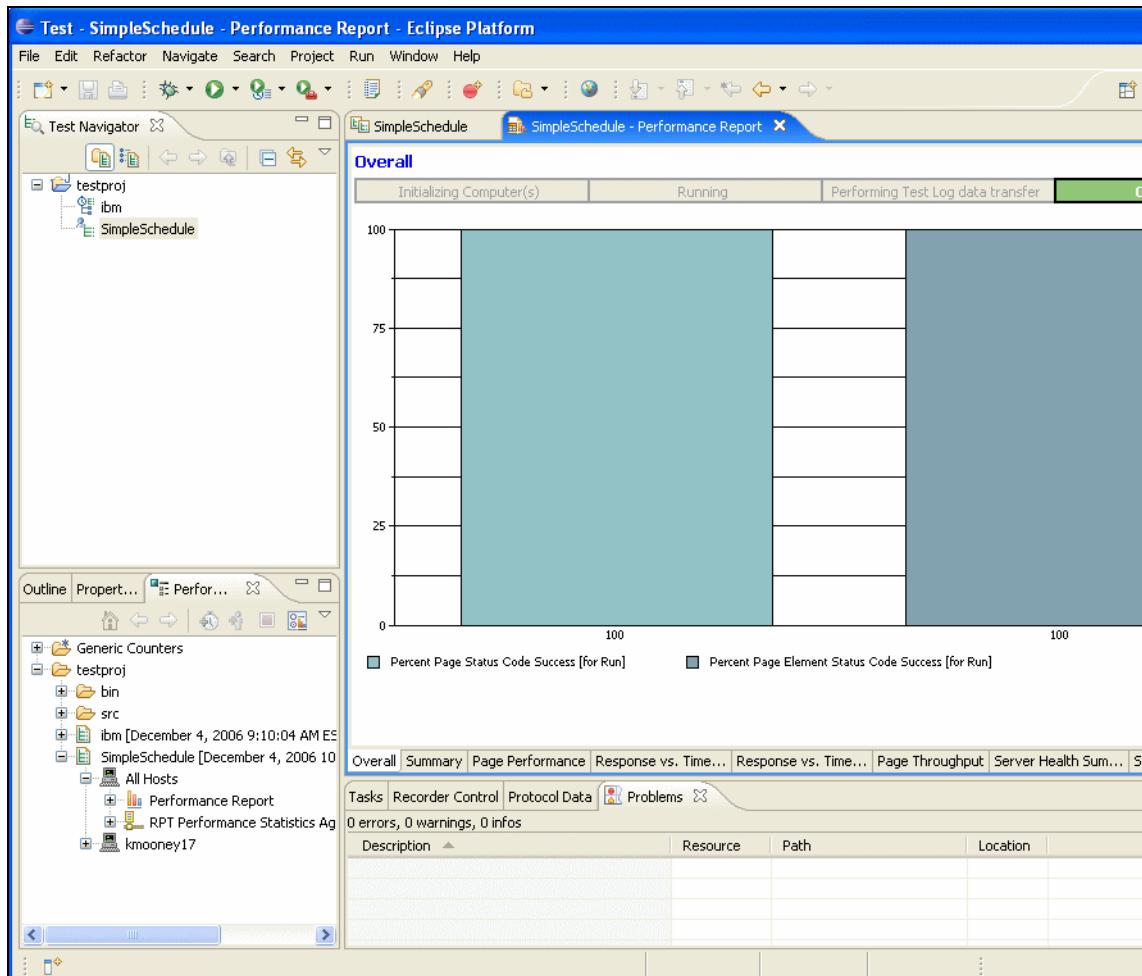


Figure 5-2 Playback status of a schedule

5.3 Playback status

As mentioned in the previous section, the Overall Status window shows the state of the schedule playback. There are four main transitional states:

Initializing computers

In this state, RPT deploys the schedule and tests to the computer or computers that will execute them.

Running / adding users

In this state, RPT is playing back the schedule.

Performing test log data transfer

During this state, the computers executing the test send detailed test playback information back to the workbench where the playback was launched.

Complete

Schedule playback has finished.

More details about schedule playback can be viewed from the Summary window. During playback, select the Summary tab to display the Summary window (Figure 5-3).

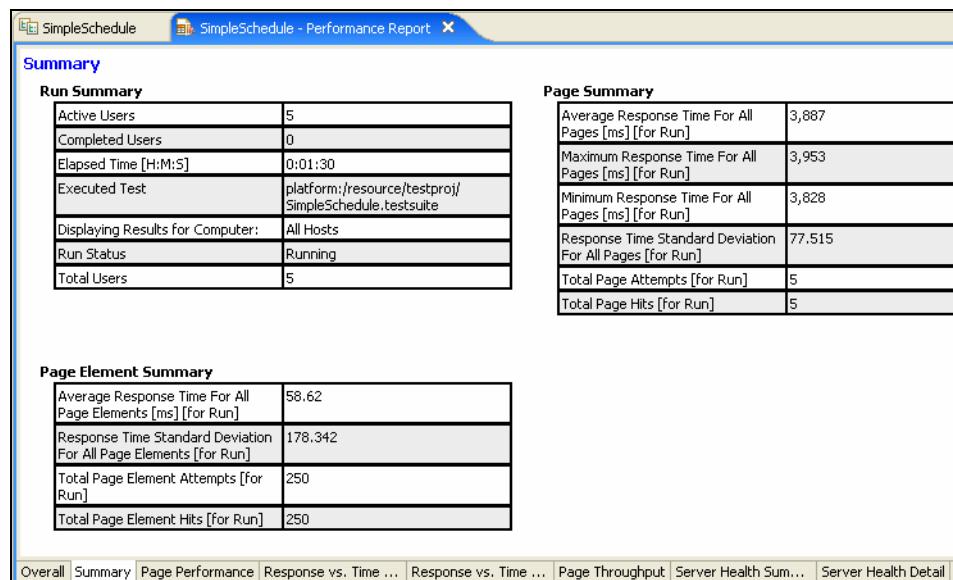


Figure 5-3 Schedule playback summary

5.4 Monitor and control

During playback it is possible to modify the test log level, add users, or stop the test run.

5.4.1 Statistics interval

The rate at which information is updated in the Schedule Performance Report windows, including the Summary Window, is controlled by the schedule statistics sample interval. In the schedule editor, select the top item in the Schedule Contents pane. The top item should be the schedule name, for example, SampleSchedule. Then, in the right-hand side of the window under Schedule Element Details, select the *Statistics* tab.

For an intensive load test playback with many virtual users, select a larger value than the default interval of five seconds. For example, select a value of 30 or 60 seconds (Figure 5-4). By changing this sample interval, no data is lost. All of the individual response time measurements are collected and reduced to interval based statistics over whatever interval is used for data collection. The main difference is how often data points are posted to the report window during the playback. This is done based on this sample interval.

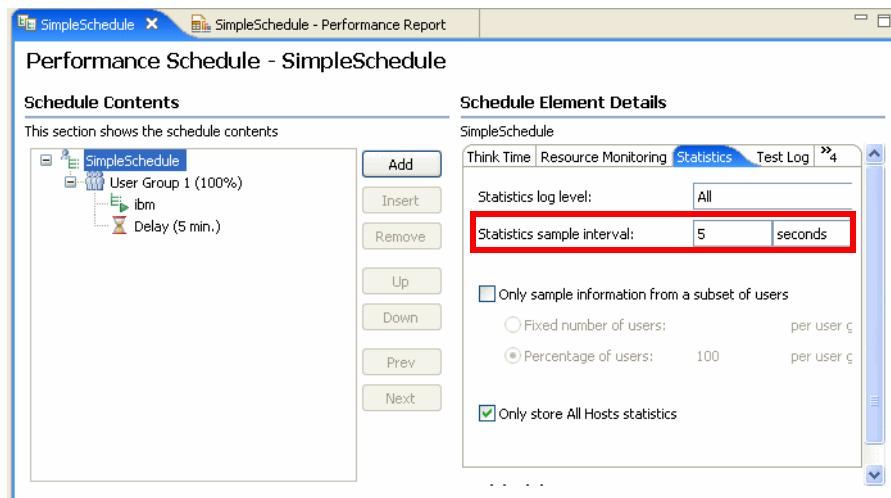


Figure 5-4 Statistics sample interval

5.4.2 Change log level

The test log provides detailed information regarding individual response times, errors, and warnings. If you suspect that schedule playback has entered a problem phase and you want to increase the current test log level, you can do so during playback. Click *Change Test Log Level* (icon), highlighted in Figure 5-5, then select the new log level.

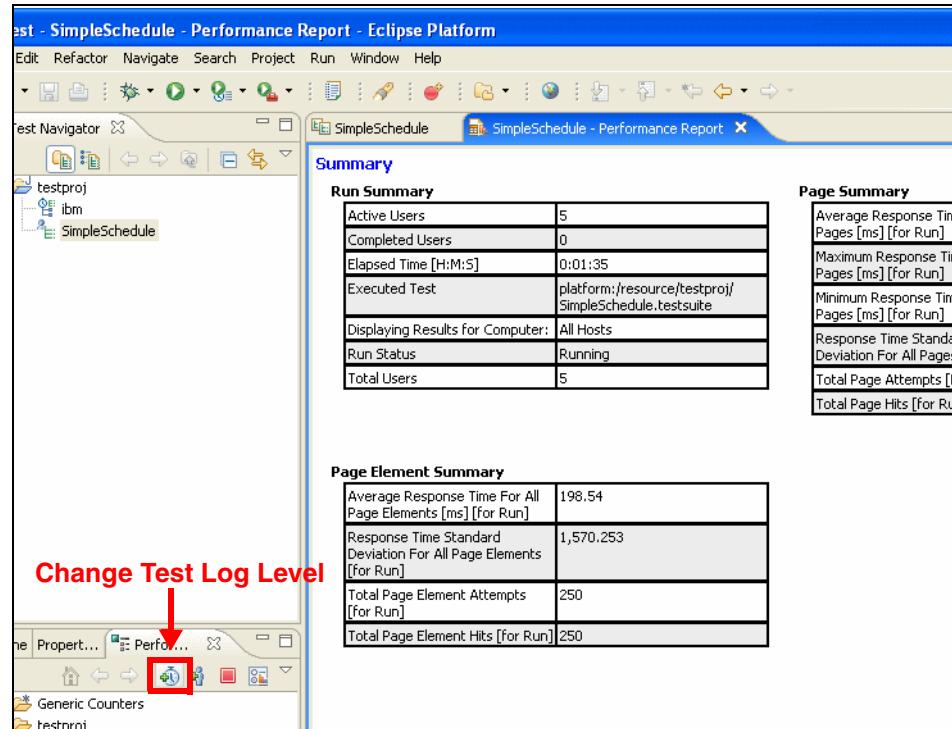


Figure 5-5 Change test log level

5.4.3 Add virtual users

During playback it is possible to increase the number of virtual users executing the schedule. Click *Add Users* (icon), highlighted in Figure 5-6, to bring up a dialog box that will prompt you for the number of users to add.

While users are being added by RPT to the schedule, you cannot add additional users. Based on the staggering option, it will take time to add the new users. If you note that the current state of the playback changes from *adding users* to *running* state, it becomes possible to click *Add Users* again to add more users.

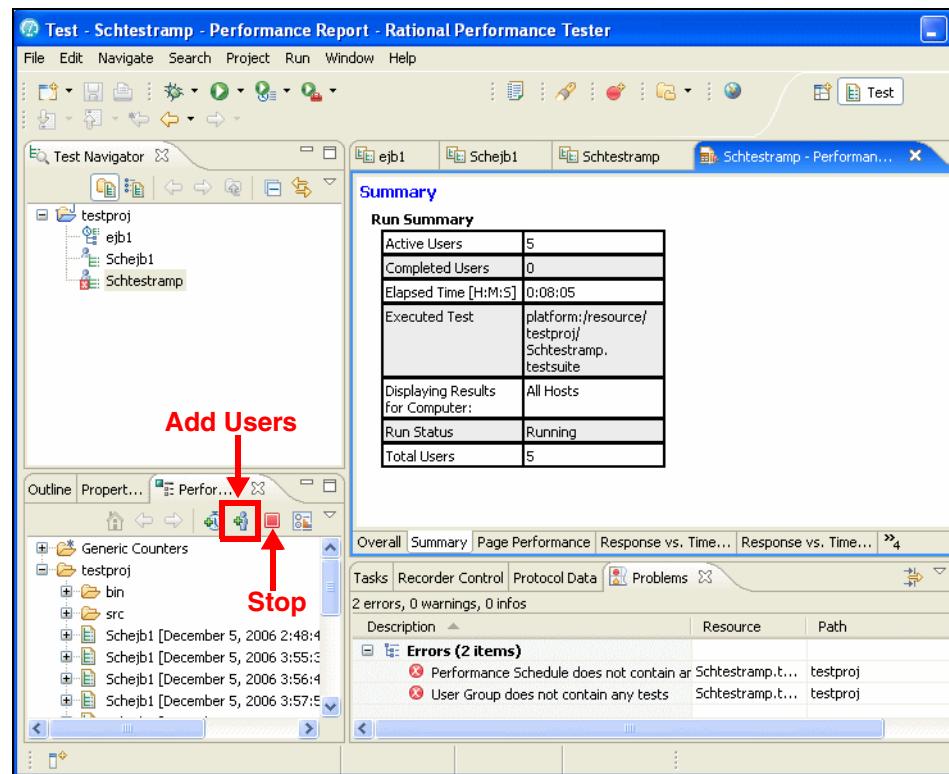


Figure 5-6 Add virtual users

5.4.4 Stop test run

Click *Stop Test Run* to stop playback (see Figure 5-6 on page 146; use the red square button next to the *Add Users* button). After you click *Stop Test Run*, you are given options for how you want to stop playback (Figure 5-7).



Figure 5-7 Stop Test Run

The options presented allow you to select how you want the stop playback to occur, and if you want test results and history information. The options are:

- | | |
|----------------------|--|
| Hard Stop | A hard stop ends playback immediately. Virtual users in the middle of executing an action are not allowed to finish the action. Select this option if you want to end now and do not need complete results. |
| Timed Stop | A timed stop allows virtual users some time to complete the action they are currently executing. An action can be a request for an image from a server or some other request/response pair depending upon the test you are running. If all virtual users do not finish within the time you specify, then a hard stop occurs. |
| Graceful Stop | A graceful stop allows users all the time they need to finish the action they are currently executing. When that action finishes, the virtual user stops, and playback stops when all virtual users have finished. |

If you clear *Collect tests results and history*, RPT will not attempt to provide this information after playback stops. If you are stopping playback because you have discovered some problem, it is possible that you have no interest in the results of this playback. Turning off this option can greatly speed the stop playback sequence.

5.5 Debugging

This section provides suggestions for dealing with common playback problems and presents tools you can use to help answer the question “What went wrong?”

5.5.1 Problem determination log

The problem determination log is a tool primarily designed to help RPT technical support personnel identify the source of problems if you call for support. The levels that control how much information is logged are available using the Schedule editor.

Collecting problem determination log data is expensive in terms of CPU and disk space. For normal operation, do not increase the level beyond the default of Warning and do not sample more users than the default of five.

The data collected is stored on each agent (location) computer executing the schedule. It is not transferred to your workbench (UI) after playback completes. Therefore, to view the problem determination log, you must logon to each agent computer and view it there.

A maximum of 10 problem determination logs, each containing up to 250 MB of data, can be collected during playback. The files will be named CommonBaseEvents00.log to CommonBaseEvents90.log. The most recent file is always CommonBaseEvents00.log. Logs from previous playbacks might also be present. Sort these files by date if necessary.

The log messages in these XML files are prefaced with `msg=`. Look for keywords such as *exception* or *error* if you are experiencing problems.

5.5.2 Launch debugging

The goal of launching is to reach the *running* state for playback. This section covers some common pitfalls, shows how to recognize them, and provides suggestions for recovery.

Detailed launch information

Any launch problem can be better understood if you have more information during the launch. To get more information, increase the logging level for launch and view the error log. Follow these steps:

- ▶ Open the logging preferences. Select *Window* → *Preferences* → *Logging*, then select the *Loggers* tab.

- ▶ Select the Logging Level window for com.ibm.rational.test.common.schedule.execution and set the level to **ALL**.
- ▶ Select the Logging Level window for com.ibm.rational.test.lt.execution and set the level to **ALL**.
- ▶ Click **OK** to save these changes.
- ▶ Open the Error Log view by selecting *Window → Show View → Error Log*.

The Error Log view should appear near the bottom of the RPT workbench.

Important: The Error Log Filters limit viewable events to 50 at a time. You can remove this restriction for viewing after playback stops. **Leave the limit of 50 viewable events during playback, or the workbench performance could be severely impaired.**

No local Agent Controller

This problem can occur if there is an installation problem, possibly related to mixing RPT with other Rational products (Figure 5-8).



Figure 5-8 No local Agent Controller

On Windows machines, there should be an ACWinService process; and on Linux machines, there should be an RAServer process running.

Recovery

On Windows, select *Start → Run → cmd* and execute the following command:

```
net start "IBM Rational Agent Controller"
```

On Linux, find the directory AgentController/bin under the path where RPT was installed and execute the following command:

```
./RAStart.sh
```

No remote Agent Controller

The Agent Controller is a process that RPT communicates with to coordinate and direct the playback of your schedule on agent (location) computers. For efficiency and fidelity of response times, it is an RPT best practice to execute the RPT workbench UI on one computer and drive test playback from agent computers.

If the following message (Figure 5-9) appears, it means that RPT was unable to find the agent controller running on the remote agent computer.



Recovery

You must install Rational Performance Tester Agent on the remote computer, or if it is already installed, you must start the Agent Controller. The commands to start the Agent Controller are the same as described in “No local Agent Controller” on page 149.

Connection refused

The Agent Controller has a security feature that allows for restricting playback to the local computer only.

If one of your agent computers has an Agent Controller configured for *local* playback only, you will receive the message shown in Figure 5-10.



Recovery: Windows

Follow these steps:

- ▶ Select *Start → Run* and execute the command **cmd** to start a command prompt.
- ▶ Change directory to the Agent Controller bin directory depending on where RPT was installed. For example:
`cd "Program Files\IBM\SDP70Shared\AgentController\bin"`
- ▶ Stop the Agent Controller:
`net stop "IBM Rational Agent Controller"`
- ▶ Run the configuration utility:
`setconfig`
- ▶ Take defaults by pressing *Return*, but specify network access mode ALL.
- ▶ Start the Agent Controller:
`net start "IBM Rational Agent Controller"`

Recovery: Linux

Follow these steps:

- ▶ Start a terminal shell window.
- ▶ Change directory to the Agent Controller bin directory depending on where RPT was installed. For example:
`cd /opt/IBM/SDP70Shared/AgentController/bin`
- ▶ Stop the Agent Controller:
`./RAStop.sh`
- ▶ Run the configuration utility:
`./SetConfig.sh`
- ▶ Take defaults by pressing *Return*, but specify network access mode ALL.
- ▶ Start the Agent Controller:
`./RAStart.sh`

5.5.3 Firewalls and virus detection software

General playback problems or connection problems can be caused by firewalls and virus detection software. When no solution is obvious, try playback with firewalls and virus software temporarily disabled.

On the workbench computer, RPT has to be added as an exception for the firewall. On the remote agent (location) computers, the Agent Controller process has to be added as a firewall exception.

5.5.4 Test run aborted

This message appears in a window as shown in Figure 5-11.

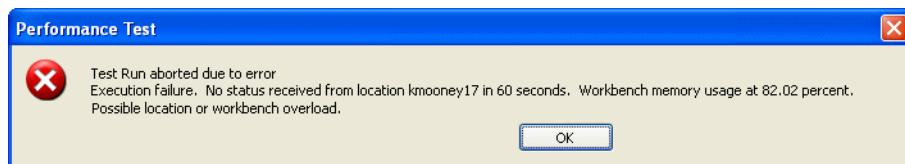


Figure 5-11 Test run aborted

What the message means is that schedule playback ended because the workbench lost communication with one or more of the agent computers.

Recovery

Go to the agent computer listed and try to determine what happened:

- ▶ Is there a playback Java process running? If so, it could indicate the origin of the problem is with the workbench, which is addressed below. You will have to kill this Java process and any related typeperf or vmstat processes.
- ▶ View the problem determination log as mentioned previously, looking for any errors or exceptions.
- ▶ Perform a search for a recent javacore.* file on the system if you suspect the Java playback process ended abnormally. The first few lines of the core file might indicate the source of the problem.
- ▶ Try again and watch the memory size of the playback process. If the process is consistently running at its max heap, it might not have enough memory.

If there does not appear to be a problem with the agent computer, make sure the workbench has sufficient memory. Try increasing the workbench heap. Or, try reducing the level and amount of execution history.

5.5.5 Out of memory error

This problem is often difficult to catch as the agent computer performs very poorly when it happens. Most often it is diagnosed using a javacore.* file. When it occurs, try increasing the max heap for the agent. If you have already done that, then another agent computer will be needed to meet the load requirements.

5.5.6 Too many files open

Look for this message when there are problems with Linux playback. If it appears in an agent's problem determination log, the open file descriptor limit per process is being reached. The super user must increase this value from the default of 1024 using the **ulimit** command prior to starting the Agent Controller.

5.5.7 Address in use

This message can appear in Windows problem determination log files. It is usually part of a general connect exception.

Recovery

Increase TCP/IP ports. By default, Windows limits the number of available ports to between 1000 and 5000. You can increase this number up to 65000. Be very careful if you modify the registry. Add the following key to the registry to increase the number of available ports:

```
HKEY_LOCAL_MACHINE → SYSTEM → CurrentControlSet → Services → Tcpip →  
Parameters → MaxUserPort
```

Set this dword to 65000 and reboot. The MaxUserPort does not exist, generally, but can be added.

If your workload involves looping, look at where the loop is specified. Loops with schedule result in new connections for each loop iteration. Loops within tests re-use connections across iterations.

5.6 Command line interface

An alternative method of playback is to use RPT's command line interface. This form of playback is provided using the **cmdline.bat** (Windows) or **cmdline.sh** (UNIX) script. Look for these scripts within your RPT installation.

You must change to the directory where the scripts reside. To get to the default location under Windows, you would issue the following two commands:

```
cd c:\program files\ibm\sdp70shared\plugins  
cd com.ibm.rational.test.lt.cmdlineexecute_7.*
```

Run the **cmdline.bat** command to display the options for using command line, for example:

```
cmdline -eclipsehome "c:\program files\ibm\sdp70"  
-plugins "c:\program files\ibm\sdp70shared\plugins"  
-workspace d:\test-workspace -project testproj -schedule Sctest
```

Executing from the command line causes RPT to perform the playback as when launched from the user interface. Results are stored in the workspace and can be viewed using the RPT user interface. There is no monitor or control available when executing a schedule using the command line interface.



Scheduling tests

This chapter describes what a performance schedule is and how to create schedules.

We cover the following topics:

- ▶ Schedule overview
- ▶ Creating schedules
- ▶ Schedule elements
- ▶ Controlling schedule behavior
- ▶ Advanced scheduling topics

6.1 Schedule overview

A performance test schedule allows the tester to execute a realistic workload against an application as defined in the workload analysis document. The workload analysis document sets a specific set of user transactions, data, load levels, and other attributes executing against the system under test. See Chapter 2, “Performance testing process” on page 13 for more information on creating a workload analysis document.

A performance test schedule is the *engine* that runs a test. The tester can emulate simple or complex workloads by defining the schedule contents and element details. For example, a schedule can:

- ▶ Group tests under user groups, to emulate the actions of different types of users
- ▶ Set the order in which tests run, for example, sequentially, randomly, or in a weighted order
- ▶ Set the number of times each test runs
- ▶ Run tests at a certain rate
- ▶ Run user groups at remote locations

6.2 Creating schedules

To create a new schedule, select *File* → *New* → *Performance Schedule*, select the parent folder where you want the schedule located, enter the name of the schedule, and click *Finish* (Figure 6-1).

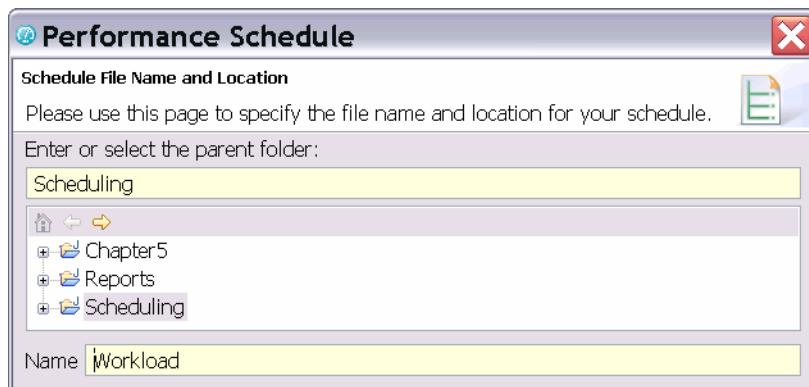


Figure 6-1 New Schedule Wizard

You add user groups, elements, and other items in the schedule contents section of the schedule.

6.3 Schedule elements

In this section we add elements to the schedule.

6.3.1 User group overview

User groups enable the tester to group tests in a logical order to emulate the actions of different types of users. A schedule can contain one or more user groups. In our first schedule example, one user group, Customers and Browsers, has been defined (Figure 6-2).

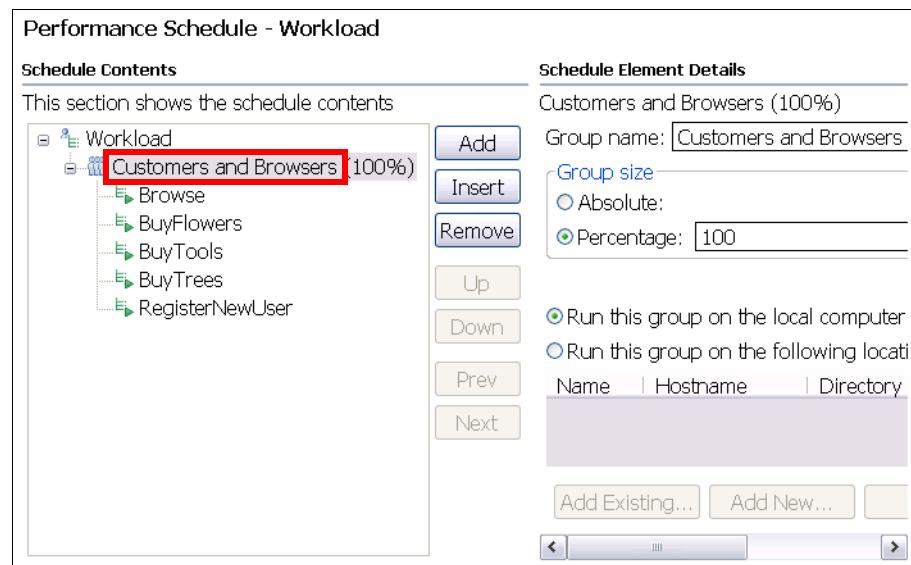


Figure 6-2 Schedule with a single user group running five sequential tests

If you run this schedule with 10 users, all 10 users would run all 5 tests sequentially. This approach does not give the tester much control over the execution. Another approach is to define a schedule with two or more user groups, where each user group represents a type of user of the system (as defined in the workload model). In our second schedule example, two user groups have been defined: Customers and Browser, which represent the types of users of the system (Figure 6-3).

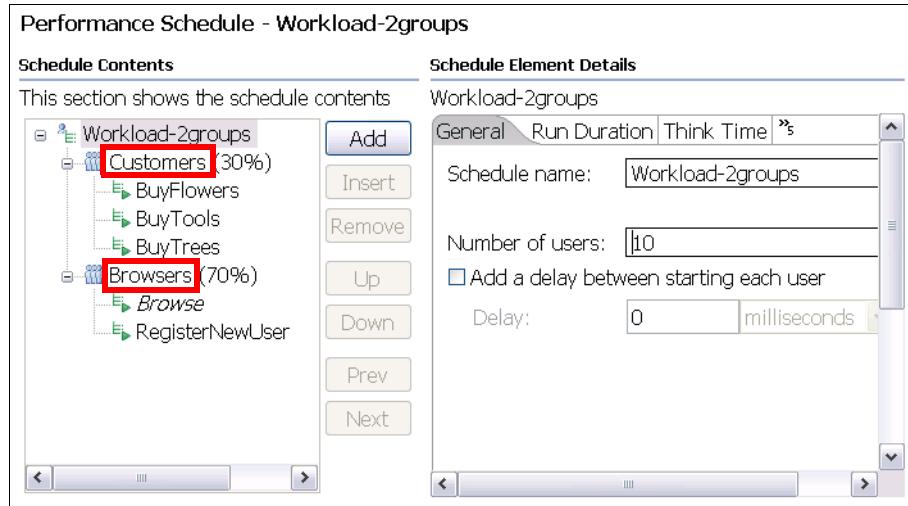


Figure 6-3 Two group schedule with a 30/70% split of users doing different tests

If you run this schedule with 10 users, three users are assigned to the Customers group and seven users are assigned to the Browsers group. When the run starts, the three customer users and seven browsers users start in parallel. You would have three customers each running three tests sequentially and seven browsers each running two tests sequentially. This is a more realistic test because each user group contains tests that represent the actions that they do, and the proportions of the user groups (70% and 30%) represent the proportions of the users on the system.

6.3.2 Adding elements to schedule

A schedule needs only one user group and one test to run. However, to accurately represent a workload, you should add other elements to the schedule. Table 6-1 lists the various schedule elements that can be added.

Table 6-1 Mapping workloads to performance schedules

| Workload Items | Schedule Element | Details |
|-----------------------|-------------------|--|
| User Types | User Groups | <ul style="list-style-type: none"> ► Percentage ► Fixed Number |
| Tasks or transactions | Performance Tests | <ul style="list-style-type: none"> ► Order ► Iterations |
| Pause in activity | Delay | <ul style="list-style-type: none"> ► Fixed period of time |
| Repeated activity | Loops | <ul style="list-style-type: none"> ► Number of Iterations |

| Workload Items | Schedule Element | Details |
|-----------------|------------------|--------------------------|
| Activity rates | Paced Loops | ► Iteration rate |
| Random activity | Random Selector | ► Define Weight of items |

Adding a user group

To add a user group to a schedule:

- In the Schedule Contents section of a schedule, select the Schedule Name and click *Add*. Select *User Group*. A user group will be added to the schedule (Figure 6-4).

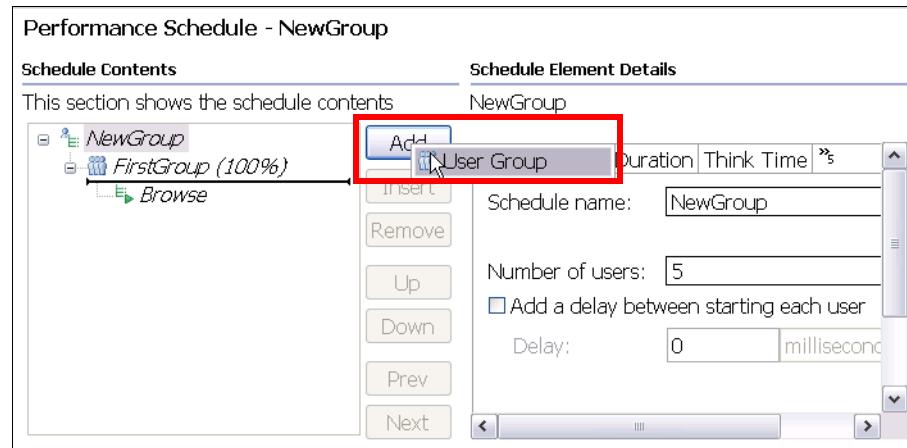


Figure 6-4 Adding a new group to a schedule

- In the Group name field, enter a descriptive name for the user group.
- In the Group Size section (Table 6-2), select *Absolute* or *Percentage*, and type the number of users or the percentage of users in the group.

Table 6-2 Group size

| Option | Description |
|----------|---|
| Absolute | Specifies a static number of virtual users. Type the maximum number of virtual users that you want to be able to run. For example, if you type 50, you can run up to 50 virtual users each time you run the schedule. Typically, you create an Absolute user group only if the group does not add a workload. For example, if you were running a test to prepare a Web site for use or a test to restore a Web site to its initial state. |

| Option | Description |
|------------|---|
| Percentage | Specifies a dynamic number of users. Type the percentage of the workload that the user group represents. In general, you assign user groups a percentage. For example, perhaps 70% of your users are browsers and 30% are customers who browse the Web site and then purchase an item. If you use percentage the total of all user groups must equal 100%. At run time the number of users entered at the schedule level will be distributed based on the percentages. For example, if 100 users was entered 70 of the users would be assigned to run the tests in the Browser group and 30 users would run the tests in the customers group. |

- ▶ Specify whether the user group will run on your computer or on another computer (Table 6-3).

Table 6-3 Run location

| Option | Description |
|---|--|
| Run this group on the local computer | The user group runs on your computer. Use this option is the workload is small or you are testing the schedule. |
| Run this group on the following locations | Generally, you should run user groups at a remote location., When user groups run from remote locations, the workbench activity on the local computer does not affect the ability to apply load. You must run a user group at a remote location in these cases: <ul style="list-style-type: none"> ▶ You are running a large number of virtual users and your local computer does not have enough CPU or memory resources to support the load. You can conserve resources by running the users on different locations, so that fewer users run on each computer. ▶ A test requires specific client libraries or software. The user group that contains this test must run on a computer that has the libraries or software installed. |

- ▶ To declare a remote location:
 - Click *Add New*.
 - In the Add Location wizard, select the project to store the location.
 - In the File Name field, type the name of the file that will contain information about this computer, and click *Next*.

Note: The data stored in the file includes information such as the host name and deployment directory. You can change this information later by opening the Test Navigator and double-clicking the file.

- In the Name field, type a descriptive name for the remote computer.
- In the Hostname field, type the IP address or the fully qualified host name of the remote computer.
- In the Deployment Directory field, type the directory on the remote computer that will store the test assets. The directory that will be created, if it does not exist, stores the temporary files that are needed during a schedule run.
- In the Operating System field, select the operating system of the remote computer and then click *Finish*.
- ▶ To add an already declared location:
 - Click *Add Existing*.
 - In the Select Location window, select the computer on which the user group will run and then click *OK*.

After you have added the user groups to a schedule, you typically add the tests that each user group will run.

Adding a test to a schedule

By adding a test to a schedule, you can emulate the action of a individual user. To add a test to a schedule:

- ▶ In the Schedule Contents section of an opened schedule, select the user group, click *Add* and select *Test*.
- ▶ Select the test from the list of existing tests and click *OK*. The test is displayed in the schedule.

Each group requires a minimum of one test.

Adding a loop to a schedule

A loop in a schedule repeats a test for a number of iterations and sets the rate for running the test. A schedule with no loops will execute each test in a user group sequentially. Loops provide more sophisticated control. To add a loop to a schedule:

- ▶ In the Schedule Contents section of an opened schedule, select the Item that will be the parent of the loop (either a user group or a test), click *Add* and select *Loop*.

- ▶ In the Schedule Element Details area, type the number of iterations that the loop will repeat.
- ▶ To maintain a set transaction rate for all schedule items that are children of this loop (Figure 6-5):
 - Select *Control the rate of iterations*.
 - At Iterations rate, type a number and select a time unit. This sets the actual rate.
 - Select or clear *Randomly vary the delay between iterations*. Selecting this check box causes the delay to vary slightly. This option models your users more accurately because the iterations are spread out randomly over a certain period of time.
 - Select or clear *Delay before the first iteration of the loop*. Selecting this check box staggers the first delay in each iteration so that you get a realistic mix at the first iteration.

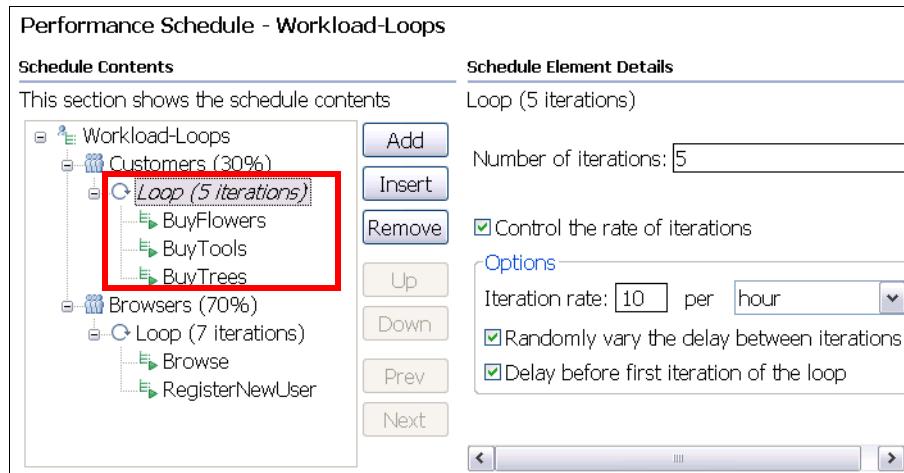


Figure 6-5 Schedule with paced loops

Note: You can also add a more granular loop within a specific test. For example, you might want to loop through specific pages or page requests.

To add a loop to a test (Figure 6-6):

- ▶ Open the test.
- ▶ Select a page or page request. The loop is inserted before the selected page or request.
- ▶ Click *Insert* and select *Loop*.

- ▶ In response to the prompt *Would you like to move selected objects into the new loop?* click Yes or No. The loop is inserted into the test, and the Test Element Details area displays the loop definition fields. If you clicked Yes, the selected items are moved under the loop.
- ▶ In the Test Element Details area, type the desired number of iterations in the Iterations field.
- ▶ Optional: Select the control the rate of iterations check box and type your preferences for the pacing rate. In specifying a number of iterations per unit of time, you set a fixed period of time for the iterations to complete. If you select *Randomly vary the delay between iterations*, the total delay is randomly distributed. If this check box is cleared, an identical delay occurs between each iteration.

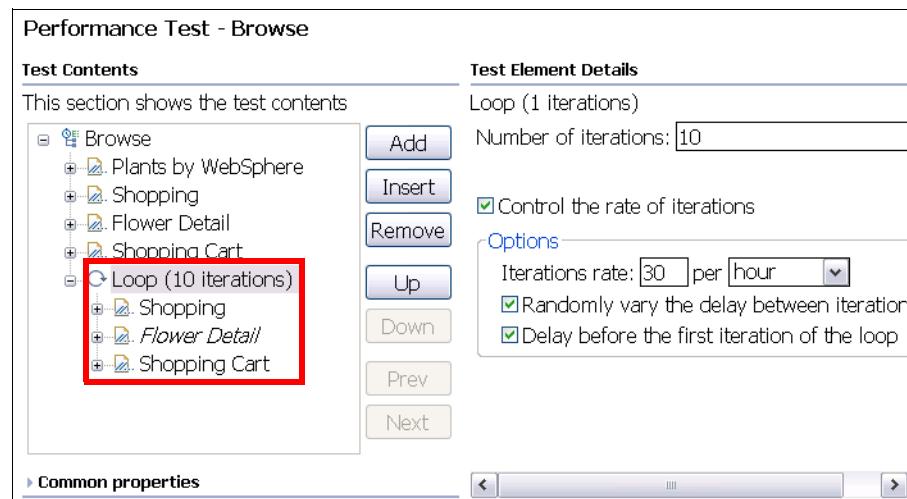


Figure 6-6 Performance test with a paced loop

Adding a delay to a schedule

A delay controls user actions. For example, you might want to slow down the initial startup of multiple users. To add a delay to a schedule:

- ▶ In the Schedule Contents section of an opened schedule, select the Item that will contain the delay, click Add and select Delay.
- ▶ In the Schedule Element Details area, type the length and the time units of the delay.

After you add a delay, you generally add the schedule items that the delay controls. The schedule items are at the same level as the delay (Figure 6-7).

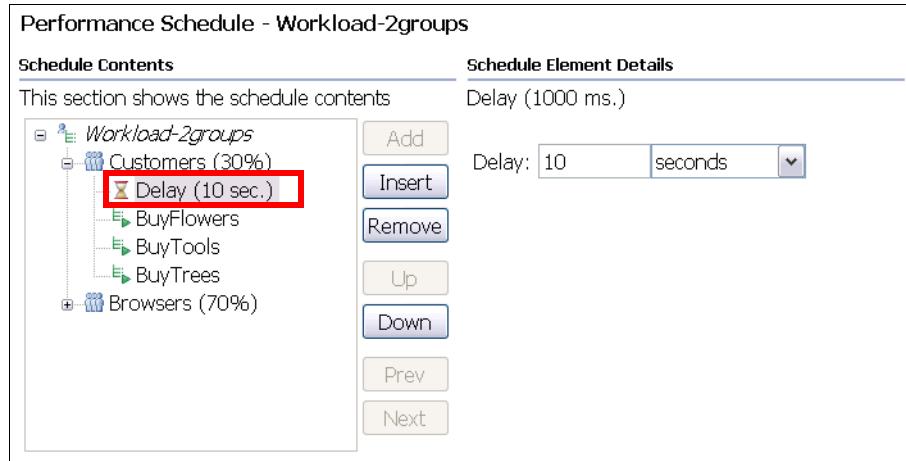


Figure 6-7 Schedule with a delay

Adding a random selector to a schedule

Random selectors are used to repeat a series of tests in a random order, emulating the varied actions of real users. For example, suppose you have a bucket that contains 10 red balls and 10 green balls. You have a 50% chance of picking a red ball and a 50% chance of picking a green ball. You pick a ball randomly, it is red. You then replace the ball in the bucket. Every time you pick a ball, you have a 50% chance of getting a red ball. Because the ball is replaced after each selection, the bucket always contains 10 red balls and 10 green balls.

To add a random selector to a schedule:

- ▶ In the Schedule Contents section of an opened schedule, select the schedule item to contain the random selector, click *Add* and select *Random Selector*.
- ▶ In the Schedule Elements Details area, add the number of iterations to loop.
- ▶ Set a weight to control the randomness of this selection:
 - Right-click the random selector and select *Add → Weighted Block*.
 - In the Weight field, type an integer. This integer shows the relative proportion that each test runs.

Assume that a random selector contains two tests: Browse and Purchase. You assign Browse a weight of 7 and Purchase a weight of 3. Each time the loop is processed, Browse has a 70% chance of being selected and Purchase has a 30% chance of being selected (Figure 6-8).

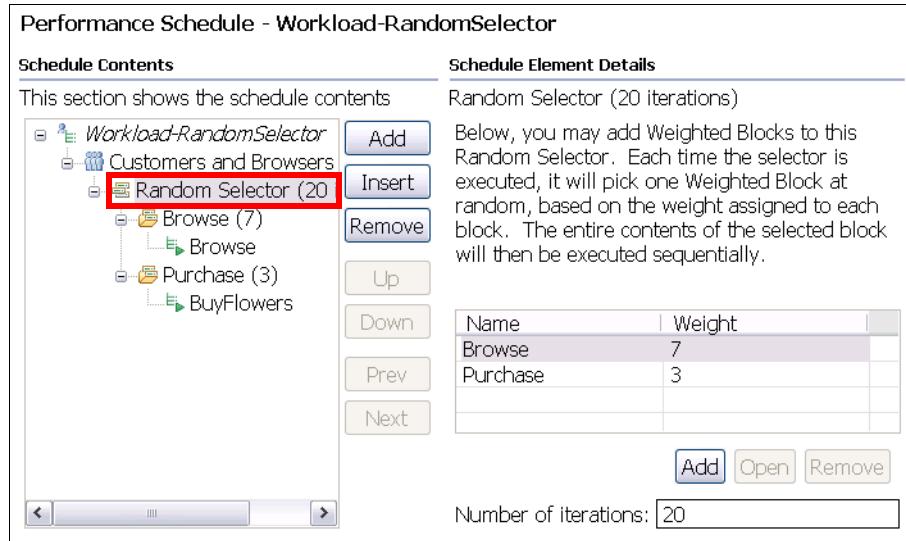


Figure 6-8 Schedule with random selector and weighted blocks

Cutting and pasting in a schedule

You can cut, copy, and paste in schedules. You can also drag and drop items.

The schedule editor supports the standard cutting, pasting and copying of schedule items using the *Edit* menu or keyboard shortcuts. Schedule items include: User groups, tests, loops, and delays. If you cut a schedule item, that item is not actually removed from the test until you perform another cut or paste operation. When you paste an item, it is displayed in italics until you save the schedule.

Ordering schedule items

To change the order of tests in the schedule, you can use the *Up* and *Down* buttons or drag and drop the schedule items.

6.4 Controlling schedule behavior

There are numerous methods to control the execution behavior of schedules includes controlling:

- ▶ Think time
- ▶ Execution rate
- ▶ User behavior
- ▶ Execution duration
- ▶ Log data collection

6.4.1 Setting think time behavior

Think time is a delay in the processing of a request to reproduce the time that a human would take to read or examine the data that is displayed from a previous user action. Think time is calculated from the time that a request is received (that is, the display is complete on the monitor) until the time that the user clicks a key or link to perform an action.

To set think time behavior:

- ▶ Open the schedule.
- ▶ In the Schedule Contents area, click the name of the schedule.
- ▶ Select the *Think Time* tab and set Modify the duration of think time delays to one of the options shown in Table 6-4.

Table 6-4 Think time

| Option | Description |
|---|--|
| Use the recorded think time | This has no effect on the think time. The time that it takes for a test to play back is the same as the time that it took to record it. For example, if you were interrupted for five minutes during recording, the same five minute think time occurs when you run the test. |
| Specify a fixed think time | Each virtual user's think time is exactly the same value - the value that you type. Although this does not emulate users accurately, it is useful if you want to play a test back quickly. |
| Increase/decr ease the think time by a percentage | You type a think time scale and each virtual user's think time is multiplied by a percentage. A value of 100 indicates no change in think time. A value of 200 doubles the think times, so the schedule plays back half as fast as it was recorded. A value of 0 indicates no delay at all. |
| Vary the think time by a random percentage | Each virtual user's think time is randomly generated within the upper and lower bounds of the percentages that you supply. The percentage is based on the recorded think time. For example, if you select a lower limit of 0 and an upper limit of 90, the think times will be between 10 percent and 90 percent of the original recorded think time. The random time will be uniformly distributed within this range. |

6.4.2 Controlling schedule execution rate

In this section we discuss setting the execution rate.

Execute at a set rate

To run a test at a set rate, you add a loop to the schedule to control the iteration rate and then add tests to the loop. The tests, which are children of the loop, are controlled by the loop.

See “Adding a loop to a schedule” on page 161 for detailed steps.

Execute at a random rate

A schedule that contains only user groups and tests will run each test in a user group sequentially. By adding a random selector to a schedule, you can repeat a series of tests in random order, thus emulating the varied actions of real users.

See “Adding a random selector to a schedule” on page 164 for detailed steps.

6.4.3 Controlling user behavior

In this section we discuss the behavior of users.

Setting number of users to run

You can set the initial number of users in a run, and then increase this number when the run starts. To set the number of users that start in the run:

- ▶ Open the schedule.
- ▶ In the Schedule Contents section, click the name of the schedule.
- ▶ In the *General* tab, set Number of Users to the number of users that you want to start.

When a schedule runs, users are assigned as follows:

- ▶ Absolute groups are allocated first. The users are allocated to each absolute user group in the order that they appear in the schedule.
- ▶ Percentage groups are allocated according to the assigned values. If a group is running on multiple remote computers, the users are divided equally among the remote locations.

For example, assume that a schedule contains the following user groups, all running on the same computer (Figure 6-9):

- ▶ If Number of Users = 2, they are all assigned to the Absolute_5 user group.
- ▶ If Number of Users = 5, they are all assigned to the Absolute_5 user group.

- If Number of Users = 10, five are assigned to the Absolute_5 user group, three are assigned to the Browsers user group, and two are assigned to the Buyers group. If the positions of the Browsers and the Buyers were reversed in the schedule, the Buyers would have three users and the Browsers would have two.

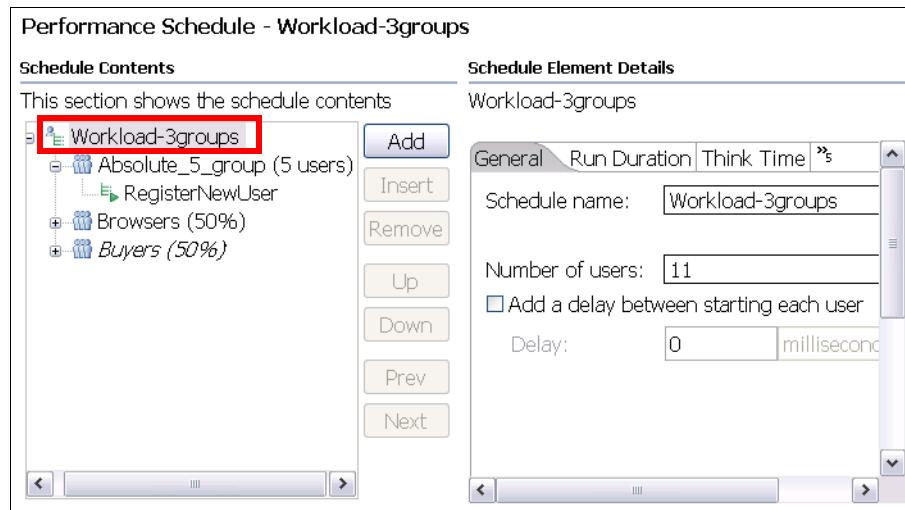


Figure 6-9 Schedule with absolute and percentage user groups

Starting users at different times

To avoid overloading the system (and causing connection time-outs), you can stagger the number of users that start rather than starting them all at once.

To stagger the starting times of virtual users:

- Open the schedule.
- In the *General* tab, Number of Users field enter the number of users that you want to start.
- In the *General* tab, select *Add a delay between starting each user*.
- For Delay, type a number and select a time unit.

Run a user group at a remote location

You can run a user group at remote locations, rather than on your local computer to prevent your workbench activity from affecting the ability to apply load.

Before you run a user group at a remote location, you must meet these requirements:

- ▶ IBM Rational Agent Controller must be installed on the remote computer.
- ▶ Firewalls on both the local computer and the remote computer must either be disabled or be configured to allow connections among the computers.

Generally, you should run user groups at a remote location. You *must* run a user group at a remote location in these cases:

- ▶ You are running a large number of virtual users and your local computer does not have enough CPU or memory resources to support this load. You can conserve resources by running the users on different locations, so that fewer users run on each computer.
- ▶ A test requires specific client libraries or software. The user group that contains this test must run on a computer that has the libraries or software installed.

Figure 6-10 shows a schedule with a user group running at a remote location.

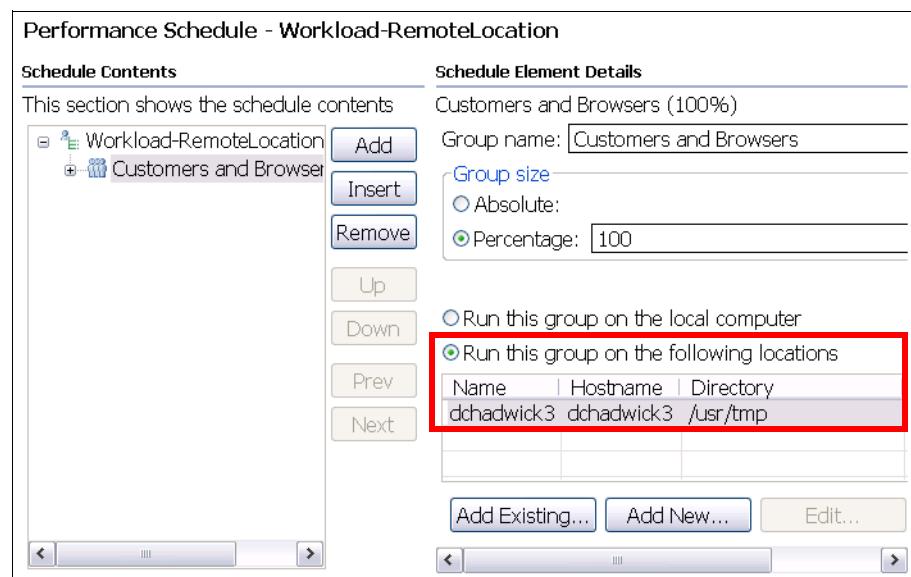


Figure 6-10 Schedule with user group running on remote location

6.4.4 Controlling execution duration

In this section we discuss how to control the duration of test execution.

Setting schedule duration

By setting the duration of a run, you can gather performance data over a specific time period.

There are two steps you must take to gather performance data over time:

- ▶ Set a schedule loop with a high iteration number (a number that will not be reached).
- ▶ Set the schedule to stop after a specific time.

To set the duration of a run:

- ▶ Open the schedule.
- ▶ Select the *Run Duration* tab and select *Stop running the schedule after an elapsed time*.
- ▶ For Stop After, type a number and select a time unit.

Controlling how a schedule stops

You can stop a schedule execution immediately (and discard results) or you can save the results so that you can run reports.

To control how a schedule stops:

- ▶ Open the schedule.
- ▶ Select the *Run Duration* tab and set the Stop Options to one of the options shown in Table 6-5.

Table 6-5 Schedule stop options

| Option | Description |
|---------------|---|
| Hard Stop | Stop the schedule run immediately. Neither the test log nor the results are saved. |
| Timed Stop | Stop the schedule run after an elapsed time. In Timeout, type a number and select a time unit. This is the default mode of stopping a schedule. |
| Graceful Stop | Allow computers at all remote locations to spend as much time as needed to fully stop on their own. The test log and the results are saved and you can report on the results. |

6.4.5 Controlling log data collection

The test log shows the events that occurred during a schedule run or test run. By setting the level of information that is collected in a schedule run, you can determine whether you receive individual response-time statistics for percentile reports and information about verification points. You can set the level of detail for each type of event (errors, warnings, and other events). By setting log levels and collecting the information from a sampling of users, you can decrease the log size.

To set the amount of information collected in the test log and the rate of sampling:

- ▶ Open the schedule.
- ▶ In the Schedule Details area, select the *Test Log* tab.
- ▶ Select the types of events you want to collect under What to Log. You can select to collect:
 - Errors only
 - Errors and Warnings
 - All Events
- ▶ For each type of event, set the log level to one of the options shown in Table 6-6.

Table 6-6 Log level options

| Option | Description |
|------------------|---|
| Schedule Actions | <p>Collects events that correspond to actions executed in the schedule:</p> <ul style="list-style-type: none">▶ The overall schedule verdict. The verdict can be one of the following values:<ol style="list-style-type: none">1. Pass indicates that all verification points matched or received the expected response. For example, a response code verification point is set to pass when the recorded response code is received during playback. If your test does not contain verification points, pass means that the connection succeeded.2. Fail indicated that at least one verification point did not match the expected response or that the expected response was not received.3. Error indicates that either the primary request was not successfully sent to the server, no response was received from the server, or the response was incomplete or could not be parsed.▶ The start and stop time of the schedule, each user group, each virtual user and each test invocation.▶ The start and stop time of each loop iteration, if loops are set in the schedule.▶ The start and stop time of each selector, if selectors are set. |

| Option | Description |
|------------------------|--|
| Primary Test Actions | <p>Typically, you set data collection at this level. Primary test actions include schedule actions plus the following actions:</p> <ul style="list-style-type: none"> ▶ Test verdict, test start, and test stop events. ▶ Loop iteration start and loop iteration stop events if loops are present in the test. ▶ Transaction start and stop events if transactions are present in the test. ▶ Page title verification points in HTTP tests. This option enables you to produce a Percentile report or to see any page title verification points that you have set. The following events are collected: <ol style="list-style-type: none"> 1. The page verdict. You see a page verdict only if a connection problem occurs or if you have set verification points. Any failures or errors are rolled up to the test verdict level. 2. The start and stop time of each page. 3. The start and stop time of each loop and the number of iterations of each loop if you have set loops within a page. 4. The start and stop time of each transaction and the duration of each transaction if you have set page-level transactions in your test. ▶ SAP screens and SAP screen title verification points for SAP tests. |
| Secondary Test Actions | <p>Secondary test actions include primary test actions plus delay events:</p> <ul style="list-style-type: none"> ▶ For HTTP tests, request level events. To collect information about response code or response size verification points that you have set, set data collection at this level of detail or greater. <ol style="list-style-type: none"> 1. The time that the first byte and last byte were sent. 2. The time that the first byte and last byte were received. 3. The character set of the response data. 4. Expected and actual values of page level verification points that you have defined. 5. HTTP think events. 6. The start and stop time of each transaction and the duration of each transaction if you have set request level transactions in your test. ▶ SAP actions for SAP tests. |
| Action Details | <p>Action details include secondary test actions, plus the following information:</p> <ul style="list-style-type: none"> ▶ For HTTP tests request and response data, for example, HTTP headers and POST data. ▶ Think times for SAP tests. |
| All | <p>All and Action Details provide the same level of data collection for both HTTP and SAP tests.</p> |

Note: All and Action Details produce large logs, especially if the tests are long or you are running a large number of users. If you select these options, set a sampling rate, rather than collecting all information from all users, which helps prevent your log from getting too large.

- To set a sampling rate (Table 6-7), select *Only sample information from a subset of users*. The number or the percentage that you select is applied to each user group. If you are running user groups at remote locations, the number or percentage that you select is distributed evenly among the remote locations.

Table 6-7 Sampling rate

| Option | Description |
|-----------------------|---|
| Fixed number of users | The number is applied to each user group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If you specify 2 for this option, two users are sampled from each group. |
| Percentage of users | The percentage is applied to each user group, but at least one user will be sampled from each group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If your sampling rate is 10%, one user is sampled from the first group and 100 users are sampled from the second group. If your sampling rate is 25%, one user is sampled from the first group and 250 users are sampled from the second group. |

You can export the statistics into a CSV file for further analysis. To export select *File → Export* and select *Test Log*.

6.5 Advanced scheduling topics

In this section we describe some advanced scheduling topics.

6.5.1 Emulating network traffic from multiple hosts

By default, when you run a schedule, each virtual user has the same IP address. However, you can make each virtual user appear as though it is running on its own host. To do this, you configure IP aliases on the host computer, and enable IP aliasing in the schedule. When you run the schedule, the network traffic will appear to be generated by multiple hosts.

Configuring IP aliasing for Windows

To make it appear that a virtual user has its own IP address during a schedule run, configure IP aliases for each Windows remote location:

- Select *Start → Control Panel → Network Connections*.
- Open the network interface that you want to associate the IP aliases with. In most cases, this is the Local Area Connection. Click *Properties*.

- ▶ Scroll down to Internet Protocol (TCP/IP) and click *Properties*.
- ▶ You must be using static IP addresses to create IP aliases on this host. Therefore, confirm that *Use the following IP address* is selected and then click *Advanced*.
- ▶ Create the IP aliases:
 - Click *Add* in the IP Addresses section of the IP Settings tab to specify the IP address of the new alias that you want to create. Make sure that the address is valid for the network and is not being used by another test.
 - Enter the IP address and the subnet mask of the new IP alias.
- ▶ After you create the IP aliases, click *OK* in each previous dialog box to complete the configuration.
- ▶ Set the schedule so that the virtual users will use the IP aliases during a run.

When you run the schedule, it will give the impression that the network traffic is being generated from multiple hosts.

You can insert custom code into your test to retrieve the runtime IP addresses of each virtual user.

Configuring IP aliases for Linux

To make it appear that a virtual user has its own IP address during a schedule run, configure IP aliases for each Linux location:

- ▶ Use the `ip` command to create an IP alias:
The following example attaches the IP address 9.37.207.29 to the eth0 network interface:

```
# ip address add 9.37.207.29 dev eth0
```

To create a large number of aliases on a Red Hat Linux platform, follow the instructions in the file:

```
/etc/sysconfig/network-scripts/ifup-aliases.
```

The following example deletes the alias:

```
# ip address del 9.37.207.29 dev eth0
```

- ▶ Set the schedule so that the virtual users will use the IP aliases during a run.

When you run the schedule, it will give the impression that the network traffic is being generated from multiple hosts.

Configuring IP aliases for AIX

To make it appear that a virtual user has its own IP address during a schedule run, configure IP aliases for each AIX® location:

- ▶ Use the **ipconfig** command to create an IP alias. To have the alias created when the system starts, add the command to the **/etc/rc.net** script.

The following example creates an alias on the **en1** network interface. The alias must be defined on the same subnet as the network interface.

```
# ipconfig en1 alias 9.37.207.29 netmask 255.255.255.0 up
```

The following example deletes the alias:

```
# ipconfig en1 delete 9.37.207.29
```

- ▶ Set the schedule so that the virtual users will use the IP aliases during a run.

When you run the schedule, it will give the impression that the network traffic is being generated from multiple hosts.

Enabling virtual users to use IP aliases

After you have configured aliases on remote computers, you set the schedule so that the virtual users can use the configured IP aliases.

Before you can enable virtual users to use IP aliases, you must:

- ▶ Configure the aliases at the remote location.
- ▶ Add the remote location to the user group.

To set the schedule so that the virtual users will use the IP aliases during a run:

- ▶ Open the schedule.
- ▶ Click the user group whose virtual users will use aliasing.
- ▶ Click *Run this group on the following locations*. The list of locations shows whether IP aliasing is enabled at that location.
- ▶ To change whether IP aliasing is enabled or disabled, click a row in the table and then click *Open*.

6.5.2 Setting problem determination level

You can set the level of information that is saved in the problem determination log during a run. By default, only warnings and severe errors are logged. Typically, you change this log level only when requested to do so by IBM Software Support.

Each remote location has a separate problem determination log, located in the deployment directory. You define remote locations and the deployment directory at each location as properties of user groups.

To set the level of problem determination logging and the sampling rate:

- ▶ Open the schedule.
- ▶ In the Schedule Details area, select the *Problem Determination* tab.
- ▶ Set Problem Determination log level to one of the options shown in Table 6-8.

Table 6-8 Problem determination log level

| Option | Description |
|-------------------|--|
| All, finest, fine | Set these options only if you are requested to do so by technical support. |
| Config | Logs status configuration messages. Configuration messages which include hardware specifications or system profiles require no corrective action. |
| Info | Logs informational messages. Informational messages that include system state require no corrective action. |
| Warning | Logs warning messages. This is the default setting. Warning messages that might indicate potential problems require no corrective action. |
| Severe | Logs critical and unrecoverable errors. Critical and unrecoverable messages interrupt normal program execution and you need to take corrective action. |
| None | Turns logging off. |

- ▶ To set a sampling rate (Table 6-9), select *Only sample information from a subset of users*. The number or the percentage that you select is applied to each user group. If you are running user groups from remote locations, the number or percentage that you select is distributed evenly among remote locations.

Table 6-9 Sampling rate

| Option | Description |
|-----------------------|---|
| Fixed number of users | The number is applied to each user group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If you specify 2 for this option, two users are sampled from each group. |

| Option | Description |
|---------------------|---|
| Percentage of users | The percentage is applied to each user group, but at least one user will be sampled from each group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If your sampling rate is 10%, one user is sampled from the first group and 100 users are sampled from the second group. If your sampling rate is 25%, one user is sampled from the first group and 250 users are sampled from the second group. |

To view the problem determination log, select *File* → *Import* → *Log File* and import the appropriate Common Base Event XML log. Check the timestamp on the log and select the one that matches the problem run.

6.5.3 Monitoring resource data

You can monitor the operating system provided resource data from any LAN-accessible system running a Windows, Linux, or Unix operating system. You can also monitor systems that are being monitored by an IBM Tivoli® Monitoring management server. In each of these cases, you specify in the schedule a location object that contains the information needed to access the data source (such as login and password data) and the resource counters from which you wish to harvest data. You can also specify parameters for how frequently you retrieve data samples from that source.

Enabling resource monitoring

From the schedule editor, you highlight the schedule root element that is the name of your schedule and then select the Resource Monitoring tab under the Schedule Element Details side of the editor. To enable resource monitoring, you simply select *Enable resource monitor* and add your data sources.

Adding resource monitoring data sources

Click *Add New* for defining a new data source or *Add Existing* to reference a pre-defined data source for resource monitoring data. For a new data source, you first fill in the host IP address or hostname and select the data source type from the left column of check boxes. A customized set of tabs is then enabled with fields to fill in that enable a connection to the data source, a tree structured set of resource counters to select, and the data collection options.

For more detailed explanation of how to configure resource monitoring, refer to Chapter 8, “Resource monitoring” on page 211.

6.5.4 Monitoring response time breakdown

You can break down the components of response time measurements by collecting additional detailed trace log data from the servers hosting your applications under test. In order for this functionality to work, you must be collecting this trace level data on every system involved in the test environment. This includes the Performance Tester workbench system, all Performance Tester Test Agent systems, and all Web servers, application servers, and optionally database servers involved in the end-to-end application transactions.

Either the Performance Tester Test Agent must be installed on all of these servers or the IBM Tivoli Composite Application Manager must be collecting the necessary transaction trace records on all of these servers. The servers must be instrumented and the data collection process active during the performance test in order to collect the response time breakdown information.

Enabling response time breakdown collection

From the schedule editor, you highlight the schedule root element that is the name of your schedule and then select the *Response Time Breakdown* tab under the Schedule Element Details side of the editor. To enable response time breakdown, you simply select *Enable collection of response time data* and select the tests on which response time breakdown data should be collected. Because this collection retrieves a large volume of trace records, the number of tests selected should be minimized to prevent potential Java heap exhaustion for large scale performance tests.

Setting logging levels

You should select *Low*, *Medium*, or *High* level for detail based on whether you desire system level, Java class level, or Java method level breakdown respectively. The quantity of data collected can grow by at least an order of magnitude with each successively more detailed level of breakdown. In addition, the amount of server overhead imposed by this data collection is successively larger with the overhead being less than 1%, about 1%, and about 10% respectively for an average J2EE™ application.

Enabling response time breakdown collection for page elements

You can also go into your tests and enable only particular page element requests so that all of the other trace data is filtered out and discarded. This permits very high load testing, while still pinpointing the breakdown information of a particular transaction across all of the servers involved.

For further detail on specifying how to collect response time breakdown information, refer to Chapter 9, “Application monitoring” on page 255.

6.5.5 Setting statistics displayed during a run

You can set the type of data that you see during a run, the sampling rate for that data and whether data is collected from all users or a representative sample.

To set the level of statistics logging:

- ▶ Open the schedule.
- ▶ In the Schedule Elements Details area, select the *Statistics* tab.
- ▶ Set Statistics log level to one of the options shown in Table 6-10.

Table 6-10 Statistics log level

| Option | Description |
|------------------------|---|
| None | No Statistics are displayed during the run and any report that depends on statistics is not generated. At the end of the run, you see only a Summary report that contains three items: the time the run took, whether the results were on the local computer (or if remote location which one) and the status of the run which is Complete. |
| Schedule Actions | Select this option if you are interested only in the number of users. Schedule actions report the number of active and completed users in the run. |
| Primary Test Actions | Select this option to limit the processing required by the workbench. Primary test actions include all schedule actions plus: <ul style="list-style-type: none">▶ For HTTP tests, HTTP page-related actions (attempts, hits, and verification points)▶ For SAP tests, SAP screens |
| Secondary Test Actions | Select this option to limit the processing required by the workbench. Secondary test actions include all primary test actions plus HTTP page element related actions. This option does not apply to SAP tests. |
| All | Provides statistics for all actions. |

- ▶ In Statistics sample interval, type a number and select a time unit. When you run a schedule, the reports show such information as response time during a specific interval, the frequency of requests being transferred during an interval and average response trend during an interval. You set this interval here.

- To set a sampling rate (Table 6-11), select *Only sample information from a subset of users*, then select one of the options in Table 6-11. The number or percentage that you specify is applied to each user group. If you are running user groups at remote locations, the number or percentage that you select is distributed evenly among the remote locations.

Table 6-11 Sampling rate

| Option | Description |
|-----------------------|--|
| Fixed number of users | The number is applied to each user group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If you specify 2 for this option, two users are sampled from each group |
| Percentage of users | The percentage is applied to each user group, but at least one user will be sampled from each group. Assume that your schedule contains two user groups. One group contains four users and one group contains 1000 users. If your sampling rate is 10%, one user is sampled from the first group and 100 users are sampled from the second group. If your sampling rate is 25%, one user is sampled from the first group and 250 users are sampled from the second group |

- Typically, you should select *Only store All Hosts statistics*. Selecting this option reduces the amount of statistical data stored, this enabling you to test a larger load over a longer period of time with significantly less memory usage. Although you will not be able to analyze data from each computer that adds to your test, this data is generally not of interest.

However, if you are running a performance test over different WANs and if you are interested in seeing the data from each remote computer, you should clear this check box. Be aware that storing statistics for each agent separately involves saving N+1 statistics models where N is the number of test agents. This has significant implications for Eclipse workbench memory usage during long test runs that might result in Java heap overflow. This mode is only recommended for short test runs where the data from multiple test agents must be reported separately.



Results analysis and reporting

This chapter describes the analysis and reporting features of the Performance Tester.

We cover the following topics:

- ▶ Introduction to analysis and reporting
- ▶ Statistical basis for reporting performance results
- ▶ Basic reports and customization
- ▶ Using reports to find time consuming application steps
- ▶ Some common scenarios for further analysis

7.1 Introduction to analysis and reporting

The primary purpose of the reporting subsystem of any performance tool is to provide the measurement data in a clearly understandable format so that conclusions can be drawn from that data. A secondary but important purpose is to determine the underlying cause for certain *slow* measurements through data analysis.

The concept of a *steady state* measurement interval is important in achieving useful measurement data. When the measurement data has been filtered to look at only the measurements made during the steady state interval, the data should be reproducible in multiple test runs and provide insight into the performance characteristics of the system.

The measurement data needs to be shown in ways that demonstrate whether the measurements represent a valid prediction of the underlying true system response time when that system is moved into production service. A discussion of the statistical basis for this assumption follows along with certain tests that can be performed on the data to ensure the validity of its prediction.

After the statistical discussion, we describe the basic reporting capabilities and how those reports can be customized to illuminate the results and analysis desired for your project. Further, we discuss several styles of data reporting and analysis used to achieve some basic results and analysis goals with reports.

7.2 Statistical basis for reporting performance results

The performance measurements of a computer system are used as a prediction of the expected system performance when the computer system is deployed. There is a body of statistical background that is behind this predictive assumption. In this section we discuss why these laboratory measurements have a bearing on the deployed system's expected performance.

7.2.1 Random arrivals yield normally distributed sample measurements

In systems theory, any causal system depends only on previous and present input values for its behavior. Any memory-less causal system depends only on its present input values for its behavior (or its output). Computer systems are certainly not memory-less systems. However, they can be treated as memory-less systems for the purpose of measuring response times if you verify that time correlation is not a factor in your results.

In a causal system, if you have random arrival rates of stimuli, then your system will emit responses with normally distributed response times for any given system function. If you categorize your responses for the same system function as samples from a single distribution, you can collect enough data samples to predict the expected response time value for the system in production. This assumes that the workload being measured is equivalent to the production workload, the system software, hardware, and data state are equivalent to the production environment, and that the measurement data is properly segmented into sample sets for the distributions of the system functions being predicted.

7.2.2 Time correlation

Time correlation is the effect of elapsed run time of the test on the measured response time taken from the system. As the elapsed time of the test increases, the response times can increase under certain internal application conditions.

For example, suppose that there is some table growing in the database that must be scanned from first to last element before a response can be sent back. The length of time scanning the table grows linearly with the number of table elements. For each transaction, if the table size grows, the response time will grow as well.

By observing the average response time at the beginning and end of the test interval and comparing them, you can determine if time correlation is a problem with your measurement data. Normally, by preloading your database tables with production levels of data, the incremental change in response time over a reasonable test interval should be very small and relative to the overall change in table sizes. This should be no more than a few percent and therefore be insignificant in your test measurements.

If you desired a more formal approach, you could apply the Student t distribution to test to see if the samples in the beginning interval and those from the ending interval had means that were statistically different. To quote from the Wikipedia entry on the Student's t-test:

http://en.wikipedia.org/wiki/Student%27s_t-test

[The Student t test is used when] a test of the null hypothesis that the means of two normally distributed populations are equal. Given two data sets, each characterized by its mean, standard deviation and number of samples, we can use some kind of t test to determine whether the means are distinct, provided that the underlying distributions can be assumed to be normal.

7.2.3 Steady state

Because of caching effects in computer systems, the system performs differently during the warm-up period at the beginning of a test interval. This is due to pre-loading frequently used items such as database index blocks or frequently accessed records into memory that are normally kept there during long term production workloads. As these data blocks are being loaded, access times can exceed the normal production response times.

These response time measurements must be excluded from the response time samples used to predict the production system response time for that operation. The interval of time when the caching and queuing effects have finally reached equilibrium is known as the steady state interval. This interval is the period from which all of the data samples should be taken that are to be used in predicting the production response times.

7.2.4 Confidence interval determination

There are many statistical tests that you can do to prove that the sample values are good enough to predict the underlying *true* response time for the system.

One compelling approach is to derive a confidence interval for the prediction of mean of the response time. Below is an example of how you might be able to build a confidence interval statement for the following data:

Mean response time (\bar{X}): 3.228 seconds

Standard deviation (σ): 0.953 seconds

Number of samples (N): 41 samples

- ▶ To derive a 90% confidence interval, you plug the numbers into the following equation where $0.90 = 1 - \alpha$:

$$\Pr[c_1 \leq \bar{x} \leq c_2] = 1 - \alpha. \quad (4.1)$$

- ▶ The Student t distribution value for 0.95, 40 is 1.684.
- ▶ This value is then plugged in to the confidence interval formula [Lilja2000]:

$$c_1 = \bar{x} - t_{1-\alpha/2; n-1} \frac{s}{\sqrt{n}} \quad (4.7)$$

$$c_2 = \bar{x} + t_{1-\alpha/2; n-1} \frac{s}{\sqrt{n}} \quad (4.8)$$

c sub 1 = 3.228 - (1.684)*(0.953) / (sqrt(41)) = 2.977
c sub 2 = 3.228 + (1.684)*(0.953) / (sqrt(41)) = 3.479

You can therefore make the statement that you have 90% confidence that the underlying mean response time is in the interval between 2.977 and 3.479 seconds.

By using some contrived data, we have derived some rules of thumb that can be used to quickly assess whether you can trust the sample data to predict the response time. If, for example, you use the following values: mean = 3.0, std. dev = 1.0, N = 20, you get the following 90% confidence interval of (2.613, 3.387). This is plus or minus 12.8%, which is marginally acceptable. So if the mean to standard deviation ratio is at least 3 and there are at least 20 data samples, you can be 90% confident that the underlying true response time is within 13% of your measured response time mean.

If your standard deviation is larger than 1/3 of the mean, you want to do additional checks to make sure you are not representing two or three different sample sets as predictors of the same functional response. It could be that some of the responses are cached or pre-fetched and others are not yielding a bimodal distribution. One way of figuring this out is to look at a frequency (histogram) chart with buckets set at ½ of a standard deviation wide. This provides a visual indication of whether the data appears to be from a Gaussian distribution (be normally distributed).

7.2.5 How much data is enough to predict the true response time

Using those same confidence interval calculations, you can derive how much data is enough to predict the true response time:

- ▶ If you want to have a 90% confidence interval for a value that is within 5% of the true response time, you can calculate how many samples you need, assuming a mean to standard deviation ratio of 3/1:

$$N = (t / (0.05 * 3))^2$$

- ▶ Begin by approximating the t value with 1.660 for n = 100.
- ▶ This yields an answer of N = 122 samples. This is the minimum number of samples required to get within plus or minus 5% of the true mean with 90% confidence, assuming a sample mean to standard deviation ratio of 3/1.

As you can see, going from a 13% to 5% confidence range raises the number of samples from 20 to 122. Many more samples are needed to obtain a tighter confidence interval. The same is true if you want to have a higher confidence value than 90%.

7.2.6 Value of percentiles for response time measurements

There needs to be a way of proving that you have enough good measurement data to predict the mean response time. However, most computer systems are designed to satisfy a majority of the users not just meeting an average requirement, where potentially half or more of the system's users might have a worse response. When the measurements are seen as coming from a Gaussian distribution, the usage of percentile data from the sample data can give you a notion of how *most users* will have a better response time than this value.

Usually a 90th or 95th percentile is used from the sample data for two purposes:

- ▶ There might be outlier data points beyond the design of the system.
- ▶ If the vast majority of users are getting equal or better response time, that is good enough.

In general, percentiles are only used for interactive response times and not for other measurements.

7.2.7 Average values for transaction throughput rates

For throughput rates, you are most interested with the maximum capacity of the system, and the only metric that matters is the average rate at which you can process the transaction or key system operation. In a large number of cases, these relatively long running operations (such as running a batch report) could yield very few sample data points.

You might have to create a special workload where more operations are done than normally, or run the workload for an extended period of time to get enough data points to gain an accurate value for the predicted transaction rate. If the run time is long enough, the elapsed time for the operation might not be very sensitive (whether it takes eight or ten minutes to run the report for example). With that in mind, you might not need many sample data points to meet the requirements of your measurement criteria.

Reference:

Measuring Computer Performance: A Practitioner's Guide, Chapter 4: Errors in Experimental Measurements, by David J. Lilja, Cambridge University Press, 2000.

7.3 Basic reports and customization

Rational Performance Tester provides counters to enable testers to analyze system response time in a variety of ways and for a variety of test targets. In the case of HTTP, response time test targets include tests, pages, page elements, and user defined transactions.

Response time counters for each of these targets are as follows:

- ▶ **Average [for Interval]**—Indicates the average response time of a target as measured across a single sample interval (Figure 7-1).

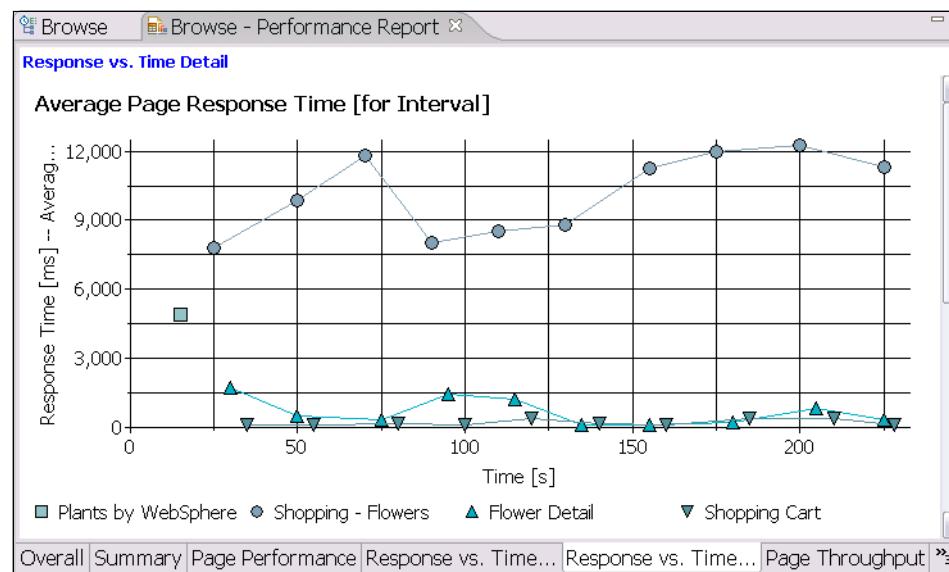


Figure 7-1 HTTP Performance Report: Response vs. Time Detail tab

- ▶ **Average [for Run]**—Indicates the average response time of a target as measured across the entirety of the test run (Figure 7-2).

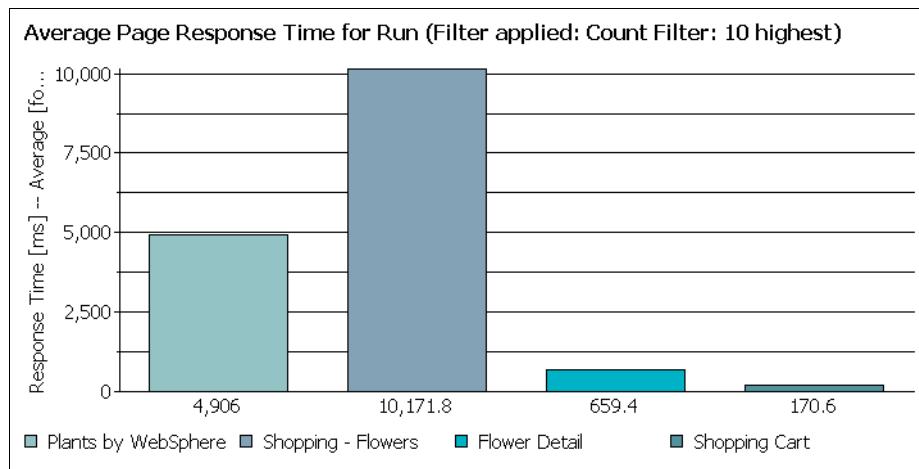


Figure 7-2 HTTP Performance Report: Page Performance tab

- ▶ **Standard Deviation [for Interval]**—Indicates the standard deviation of the response time measurements which were used in calculating the Average [for Interval] response time value. This counter is not typically included in standard Performance Tester reports but can easily be added to standard reports or included in custom reports. Report customization is discussed in a later topic.
- ▶ **Standard Deviation [for Run]**—Indicates the standard deviation of the response time measurements which were used in calculating the Average [for Run] response time value (see tables included in Figure 7-1 and Figure 7-2).
- ▶ **Average Response Time for All <target> [for Interval]**—Indicates the average response time for all like targets (i.e. all pages in an HTTP test run) as measured across a single sample interval (Figure 7-3).
- ▶ **Average Response Time for All <target> [for Run]**—Indicates the average response time for all like targets (that is, all pages in an HTTP test run) as measured across the entirety of the test run (Figure 7-3).
- ▶ **Response Time Standard Deviation for All <target> [for Interval]**—Indicates the standard deviation of the response time measurements of all like targets (that is, all HTTP pages in a test run) across a single sample interval. This counter is not typically included in standard Performance Tester reports.
- ▶ **Response Time Standard Deviation for All <target> [for Run]**—Indicates the standard deviation of the response time measurements of all like targets (that is, all HTTP pages in a test run) across the entirety of the test run (Figure 7-3).

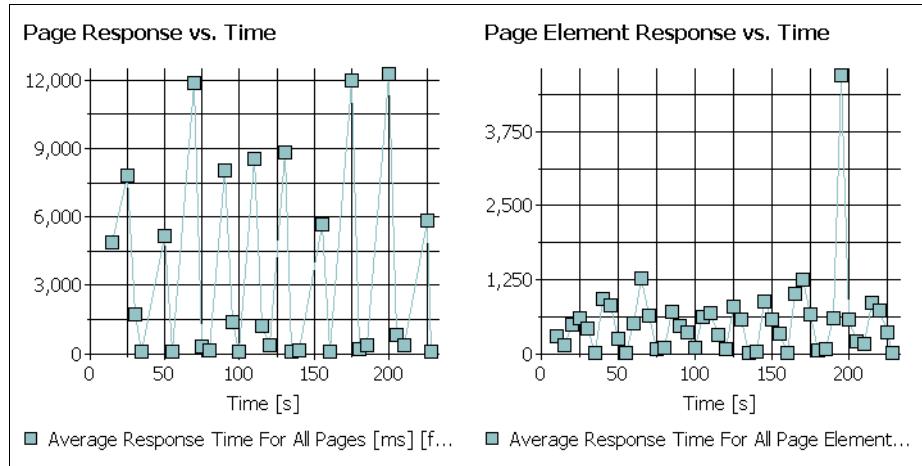


Figure 7-3 HTTP Performance Report: Response vs. Time Summary tab

In addition, some higher level targets (HTTP pages) have the following response time counters:

- ▶ **Maximum [for Interval]**—Indicates the maximum response time measurement collected for a target during a single sample interval. This counter is not typically included in the default Performance Tester reports.
- ▶ **Minimum [for Run]**—Indicates the minimum response time measurement collected for a target during a single sample interval (Figure 7-2).
- ▶ **Maximum [for Interval]**—Indicates the maximum response time measurement collected for a target during a single sample interval. This counter is not typically included in default Performance Tester reports.
- ▶ **Maximum [for Run]**—Indicates the maximum response time measurement collected for a target during the entirety of the run (Figure 7-2).
- ▶ **Maximum Response Time for All <target> [for Interval]**—Indicates the maximum response time collected for all like targets during a single sample interval. This counter is not typically included in standard Performance Tester reports.
- ▶ **Maximum Response Time for All <target> [for Run]**—Indicates the maximum response time collected for all like targets during the entirety of the run (Figure 7-4).
- ▶ **Minimum Response Time for All <target> [for Interval]**—Indicates the maximum response time collected for all like targets during a single sample interval. This counter is not typically included in standard Performance Tester reports.

- **Minimum Response Time for All <target> [for Run]**—Indicates the maximum response time collected for all like targets during the entirety of the run (Figure 7-4).

| Run Summary | | Page Summary | |
|----------------------------------|---|--|-----------|
| Active Users | 0 | Average Response Time For All Pages [ms] [for Run] | 3,707.226 |
| Completed Users | 1 | Maximum Response Time For All Pages [ms] [for Run] | 12,250 |
| Elapsed Time [H:M:S] | 0:03:48 | Minimum Response Time For All Pages [ms] [for Run] | 62 |
| Executed Test | platform:/resource/Reports/Browse.testsuite | Response Time Standard Deviation For All Pages [for Run] | 4,719.663 |
| Displaying Results for Computer: | All Hosts | Total Page Attempts [for Run] | 31 |
| Run Status | Complete | Total Page Hits [for Run] | 31 |
| Total Users | 1 | | |

| Page Element Summary | |
|--|---------|
| Average Response Time For All Page Elements [ms] [for Run] | 547.377 |
| Response Time Standard Deviation For All Page Elements [for Run] | 688.119 |
| Total Page Element Attempts [for Run] | 313 |
| Total Page Element Hits [for Run] | 313 |

Figure 7-4 HTTP Performance Report: Summary tab

In most Performance Tester tests, performance data is collected from more than one load driver. As this data is collected, it is aggregated to a single statistical model located beneath the *All Hosts* location (Figure 7-5).

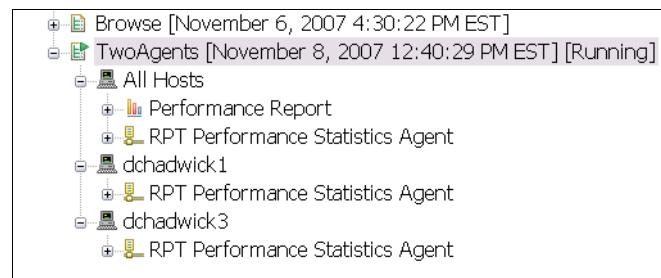


Figure 7-5 Statistical model hierarchy showing drivers and aggregate node

For most circumstances, this aggregate data is all that the tester is interested in. However, if testers are interested in analyzing response time from the point of view of a particular driver, they can elect to have that data persisted by de-selecting *Only store All Hosts statistics* on the Statistics tab of the schedule root in the schedule editor (Figure 7-6).

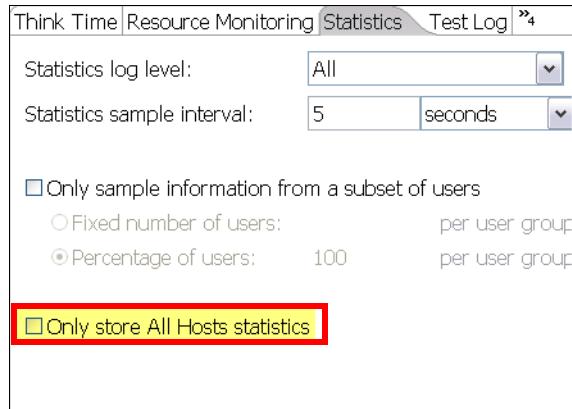


Figure 7-6 Deselecting *Only store All Hosts statistics* for driver specific analysis

After executing a test with this option deselected, the tester can open a report focused on a particular driver's data by right-clicking on the driver in the Performance Test Runs View and using the provided context menus (Figure 7-7).

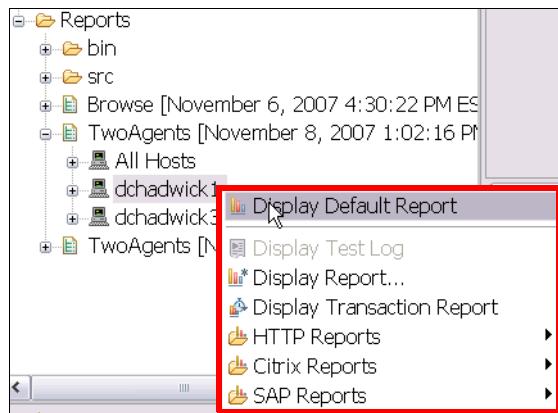


Figure 7-7 Opening a report focused on a particular driver's data

7.3.1 Correlating multiple results on one report

When analyzing performance problems and bottlenecks, it is often desirable to correlate the change in a particular counter with change in another counter. For example, a tester might be interested in the correlation between the number of active users and the response time of all page elements per interval. While default Performance Tester reports do not provide a graphic comparing these two counters, it does provide a line-chart showing the average response time for all page elements for interval (Figure 7-3 on page 189). Current user-load can easily be added to this graphic in any of three ways: Drag and drop, Add/Remove Performance Counters wizard, and Report wizard.

Drag and drop

Any counter can be added to any report graphic by selecting it in the counter in the statistical model as shown in the Performance Test Runs View and drag-and-dropping it on the target graphic. In the case at hand, the Run/Active Users/Count [for Run] counter would be dropped onto the Page Element vs. Time graphic as shown on the Response vs. Time Summary tab of the HTTP Performance Report (Figure 7-8).

When modifying a report in this fashion, be aware that the counter is always selected from the same result and location that the report is focused on. Drag and drop of a counter from non-focus results or location is useful for comparison, but creates a static link to the selected results and location that might be persisted in the report.

Subsequent uses of a report persisted in this fashion will result in always opening the statically linked result and thus will consume more memory than expected. An alternative to selecting the counter beneath the run of focus is to select the counter beneath the *generic counters* root. Generic counters have no result associated with them and thus add no static link to the modified report.

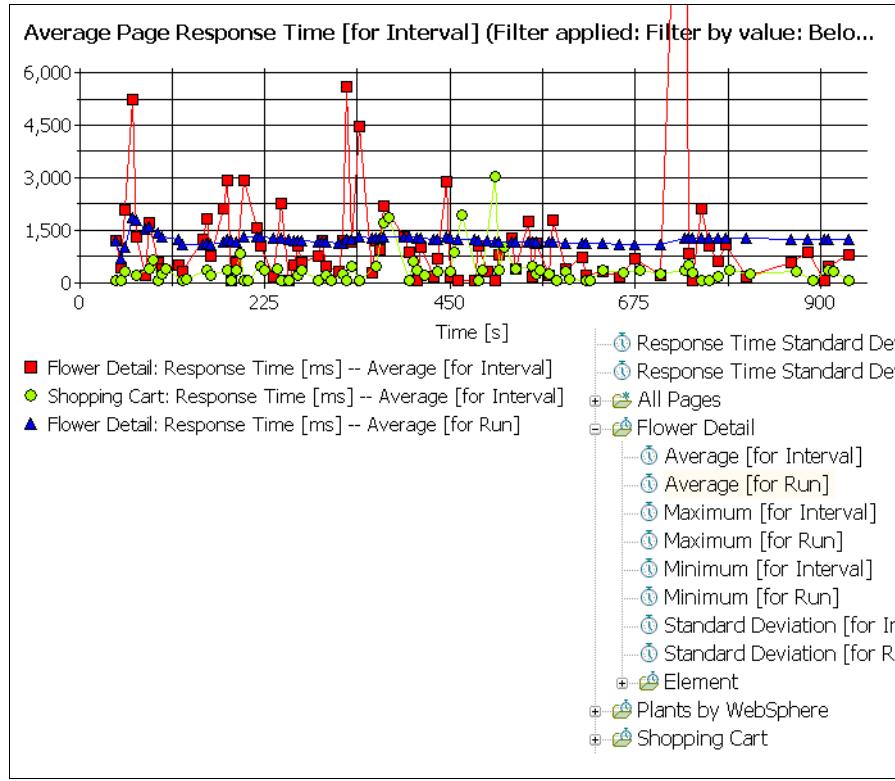


Figure 7-8 Adding a counter to a report through drag and drop from the statistical model hierarchy

Add/Remove Counters wizard

Any counter can be added to any report graphic using the counters associated add/remove counters wizard. Add/Remove Counter wizards are available in the right-click context menu of any report graphic (Figure 7-9).

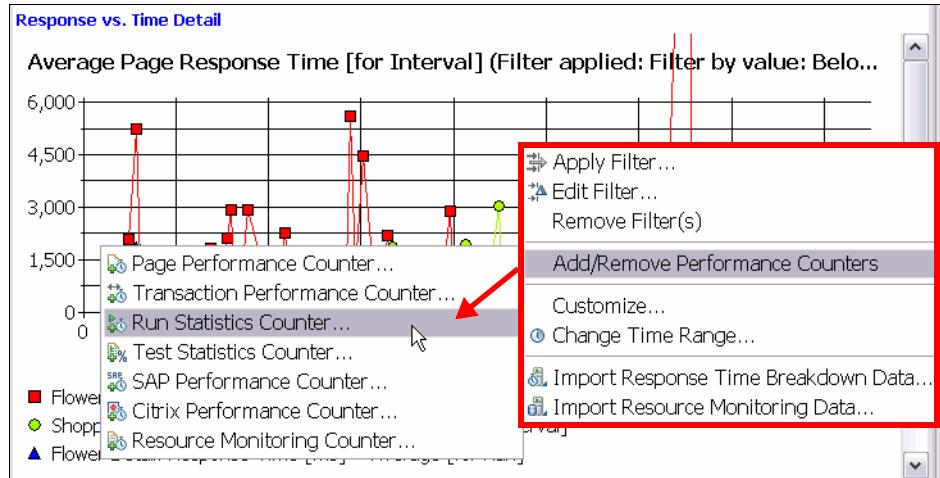


Figure 7-9 Accessing add/remove counter wizards from the report graphic context menu

Because the counter Run/Active Users/Count [for Run] is located beneath the Run model root, it can be added to the graphic of interest using the Add/Remove Run Statistics wizard. When displayed, the counter wizard will indicate any counter in its focus category that is currently displayed on the focus graphic. Counters from the category can be added or removed from the graphic by selection and de-selection from the tree control. In the case at hand, we select the *Run/Active Users/Count [for Run]* counter and click *Finish* (Figure 7-10).

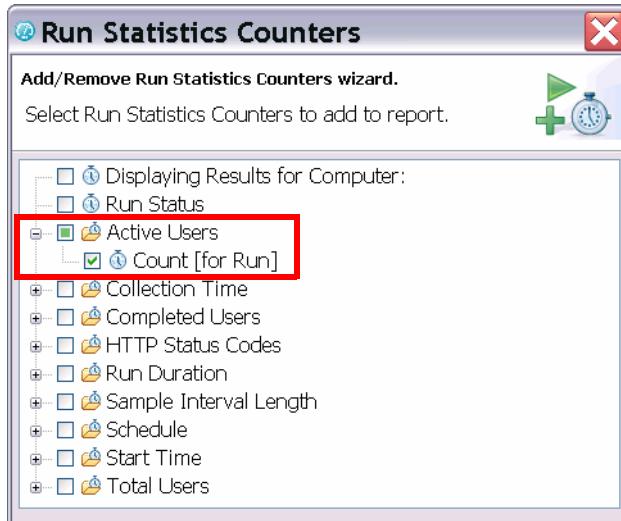


Figure 7-10 Selecting counters of interest from an Add/Remove Counters wizard

Report wizard

Along with a host of other report customizations, counters can be added to a graphic by editing the report with the Report wizard. To edit a report, select the report instance as shown in the Performance Test Runs view and use the *Edit* context menu (Figure 7-11).

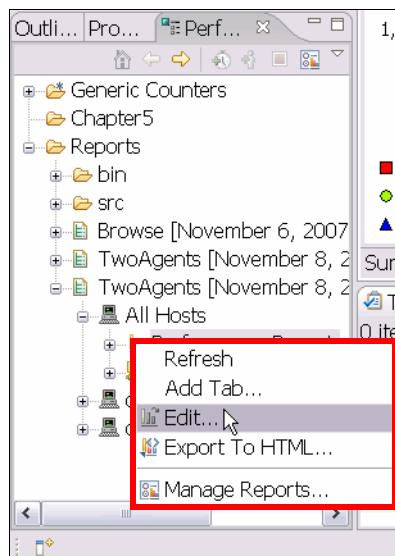


Figure 7-11 Editing a report instance from the Performance Test Runs view

The first page of the Report wizard allows testers to select the report tab they want to edit. In the case at hand, we select the *Response vs. Time Summary* tab and click *Edit* (Figure 7-12).

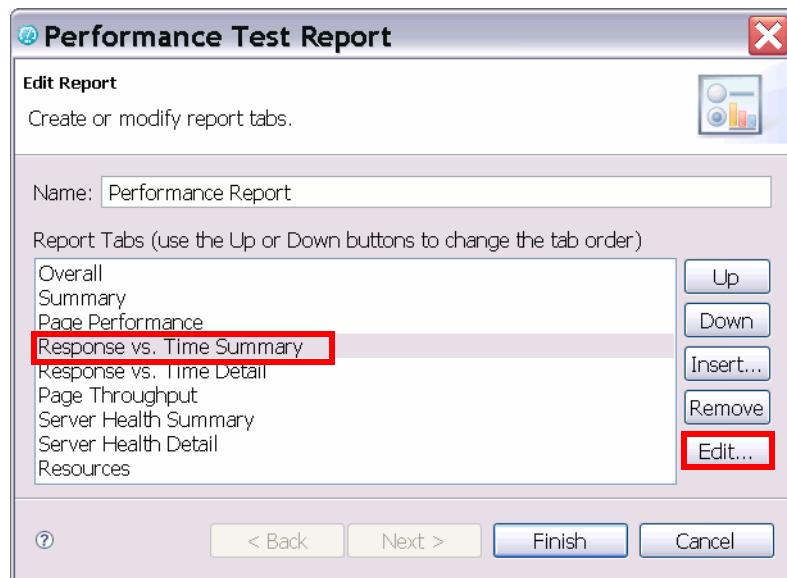


Figure 7-12 Selecting a particular report tab for modification with the Report wizard

The next page of the wizard allows the tester to change the title of the tab or to change its layout to comply with a provided template or custom layout. Because no layout or title changes are desired, we click *Next*.

The next series of pages will be pairs for each graphic on the selected report tab. The first of the pair allows title changes, graphic type changes, filter applications, and other customizations. The second of each pair allows counter additions and removals from the focus graphic. To add a counter to the Page Element Response vs. Time tab, click *Next* until the counter addition/removal page is shown for that graphic (Figure 7-13).

By default, only Generic Counters are shown in the wizard and in most cases will serve the tester's intent. To add *Run/Active Users/Count [for Run]* to the graphic, we navigate to the counter in the Generic Counters tree and click *Add*. Because no other changes are required for this report, we click *Finish*.

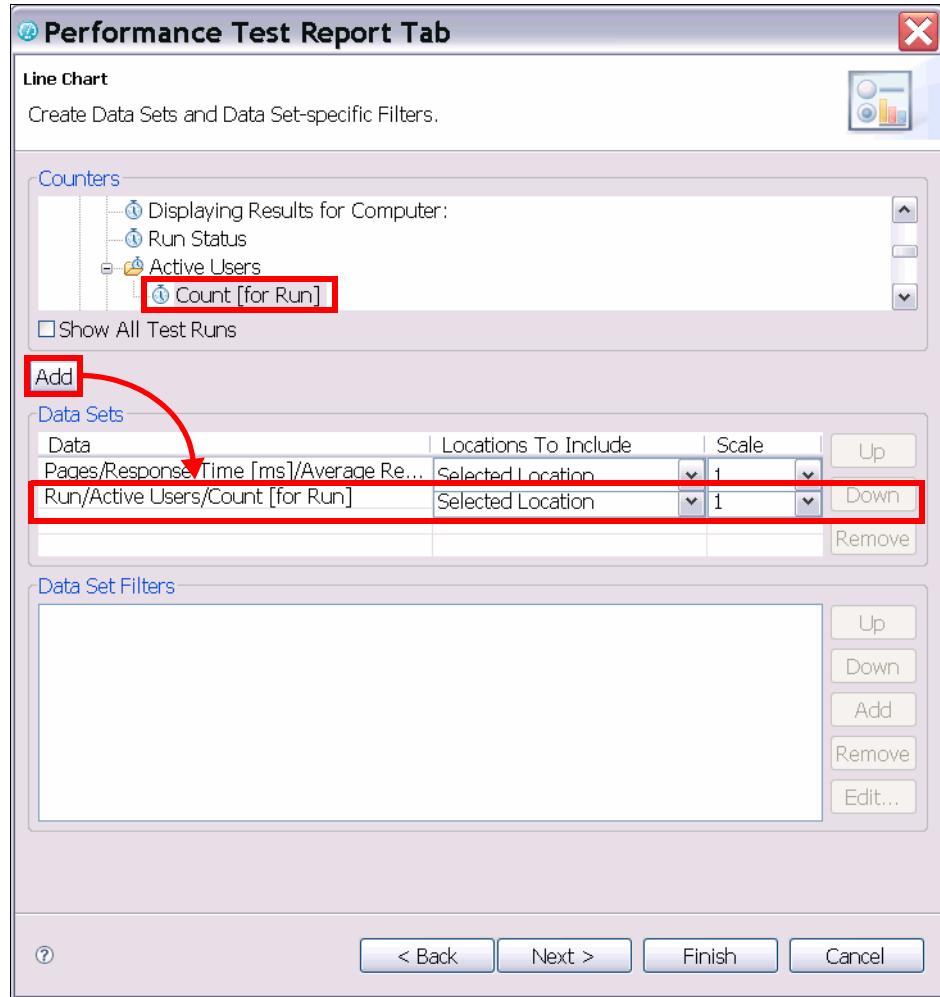


Figure 7-13 Add/remove counter page for the Response vs. Time Summary tab

After adding the Active Users counter to the Page Element Response vs. Time graphic, the report appears as shown in Figure 7-14.

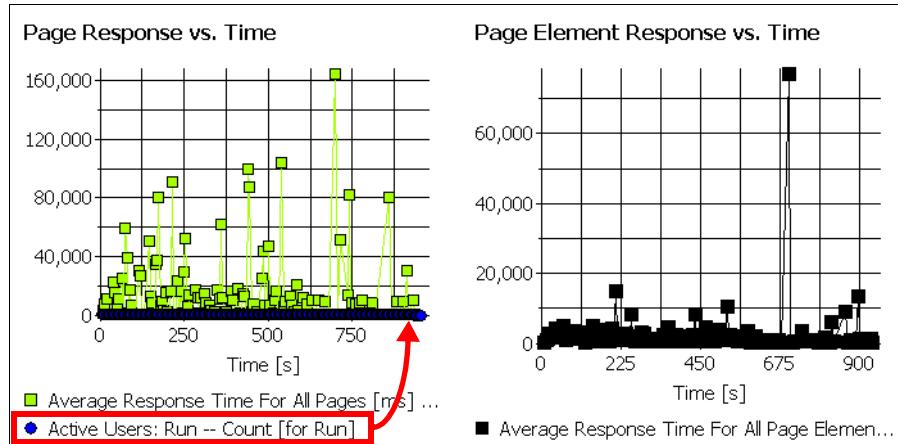


Figure 7-14 Modified Page Element Response vs. Time graphic without scaling

In many cases, the difference in average value of two counters being correlated on a single graphic causes one or more of the counters to *wash out* (appear as a flat line across the bottom of the graphic). To correct this condition, the report graphic can be edited in the Report wizard and a scale factor applied to washed-out counters (Figure 7-15).

This scale factor causes the counters' trends to be comparable to the other trends on the graph. The true value of the counter can be seen in the hover text of the scaled trend and easily interpreted mentally, because scale factors are always factors of 10.

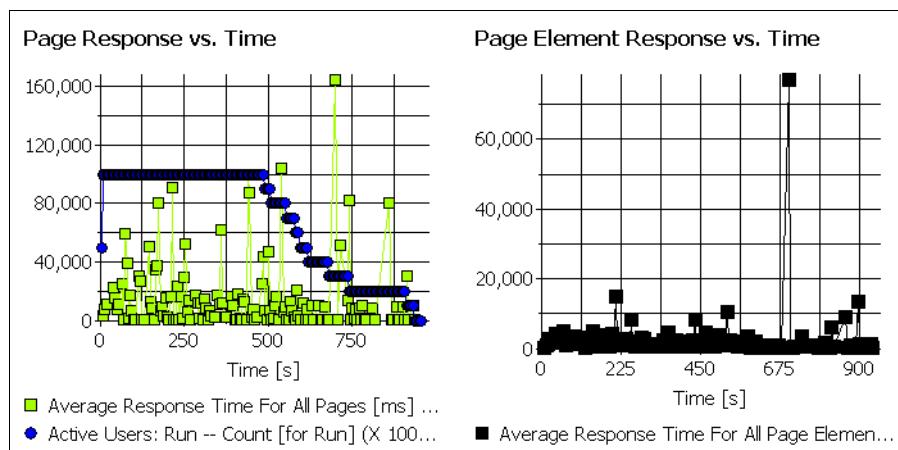


Figure 7-15 Modified Page Element Response vs. Time graphic with scaling

7.3.2 Analyzing performance within time ranges

It is often desirable to evaluate a subset of the data contained within a performance test result. One example is the analysis of a systems performance without ramp-up and ramp-down data skew. Another is the analysis of a systems performance at a particular time of day when data collection spanned the entirety of the day. This analysis is accomplished using time ranges.

A time range can be created and focused upon by right-clicking on any graphic on any open report and using the *Change Time Range* context menu item (Figure 7-16).

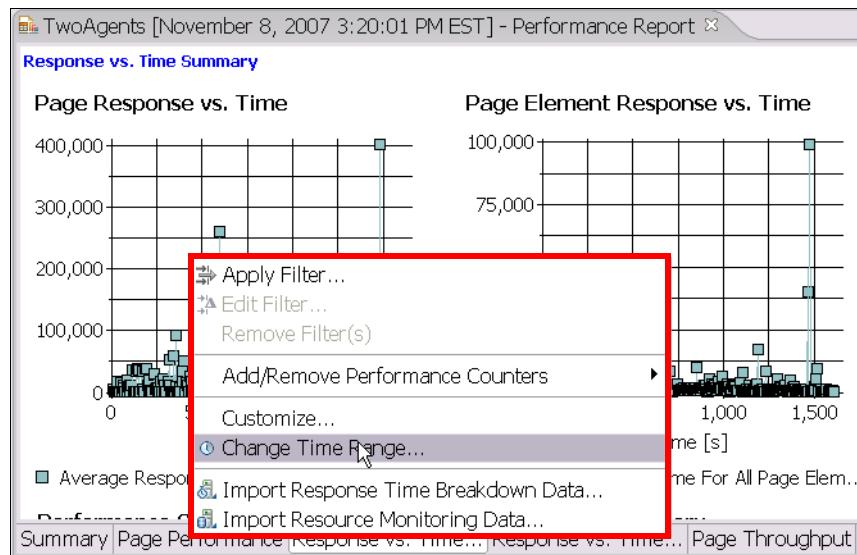


Figure 7-16 Creating a time range from a Performance Tester report

The Time Range wizard shows previously defined time ranges, including the default time range created at runtime and spanning the entirety of the run (Figure 7-17).

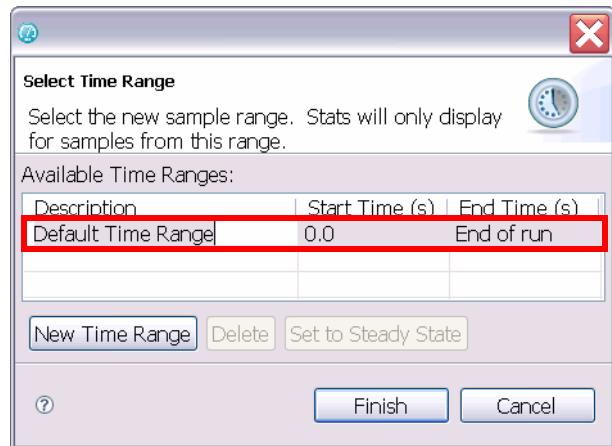


Figure 7-17 Time Range wizard

To create a new time range, click *New Time Range*. A new time range definition appears in the list where its endpoints and description can be edited. If the user is merely interested in eliminating ramp-up and ramp-down, click *Set to Steady State* for the new time range, and the endpoints of the selected time range are modified to begin at the first sample interval after the maximum number of active users is achieved, and to end at the last sample interval that contains that number of active users (Figure 7-18).

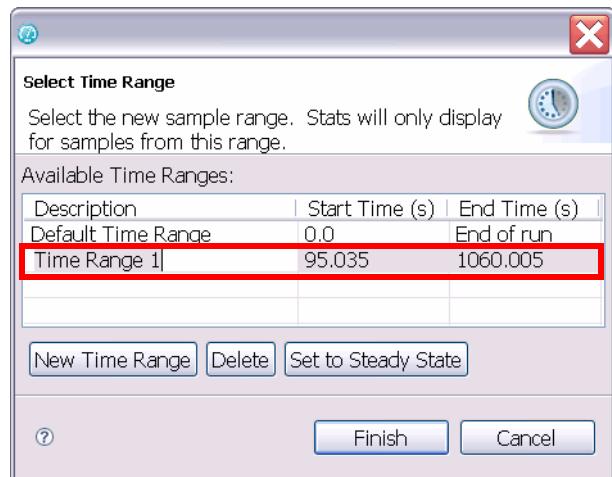


Figure 7-18 A steady state time range

To focus on the newly created time range, select it in the table and click *Finish*. A progress dialog is displayed because all statistics are re-calculated for the new time range. After generation is complete, the title of every report tab indicates that it is focused on the new time range (Figure 7-19).

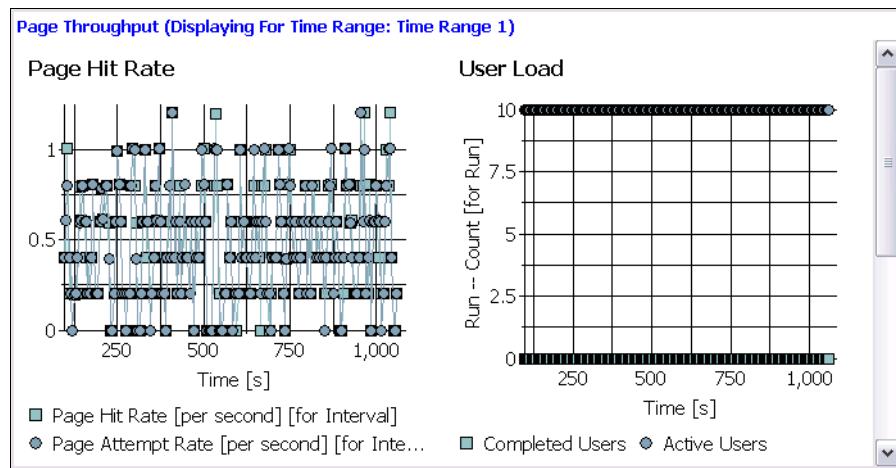


Figure 7-19 Performance Tester report focused on a custom time range

If, at a later time, testers want to change focus back to the default time range or to define a new time range, they can do so by revisiting the Time Range wizard. Until the time range focus is changed through the wizard, any analysis activity carried out on the result will be done on the focus time range.

7.3.3 Using percentile reports for response times

Service level agreements are often written with specification of maximum response for the majority of users rather than the average response time of the system. This specification is narrowed by raising the percentage of users considered as part of the majority. For example, an agreement might specify that 95% of users must experience a maximum response time of 1.5 seconds during peak operating hours when user load is expected to be 30 users.

This analysis is performed by executing a test with typical user behavior and 30 virtual users. After the result is captured, a steady state time range should be defined focusing on the 30 user load. The HTTP Percentile report is accessed by selecting a result in the Performance Test Runs view and then *Percentile Report* from the HTTP Reports sub-context menu. As shown in Figure 7-20, the first tab of the HTTP Percentile report considers all pages together and reports the slowest response time seen by 85, 90, and 95 percent of the users in the 30 user test.

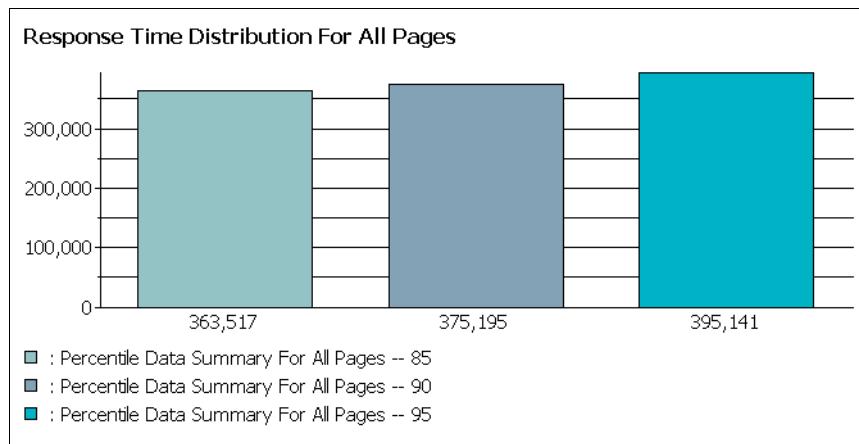


Figure 7-20 Percentile report tab showing composite percentile values

As shown in Figure 7-21, the remaining three tabs present the same type of data as the first tab but analyzed on a per-page basis.

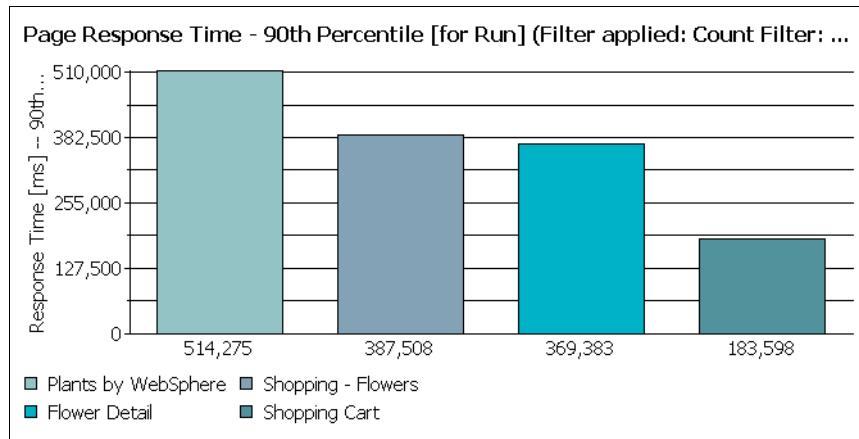


Figure 7-21 Percentile report tab showing per-page percentile value

7.3.4 Measuring transaction rates

The capability of a system is often best expressed as the maximum rate at which it can process transactions. When the maximum rate is passed, response time will increase in order for the server to continue bearing the load.

Performance Tester provides transaction rates with regard to attempts and hits for all supported transaction targets. An attempt is accomplished when a virtual user sends a request. A hit is accomplished when the system under test returns a response. Any response means that a hit does not imply a return code that would mark success such as an HTTP 200 code.

All Performance Tester protocols provide a graphic on one or more of their reports which shows the overall hit and attempt rate. For HTTP, this graphic is shown on the Page Throughput tab of the Performance Report (Figure 7-22).

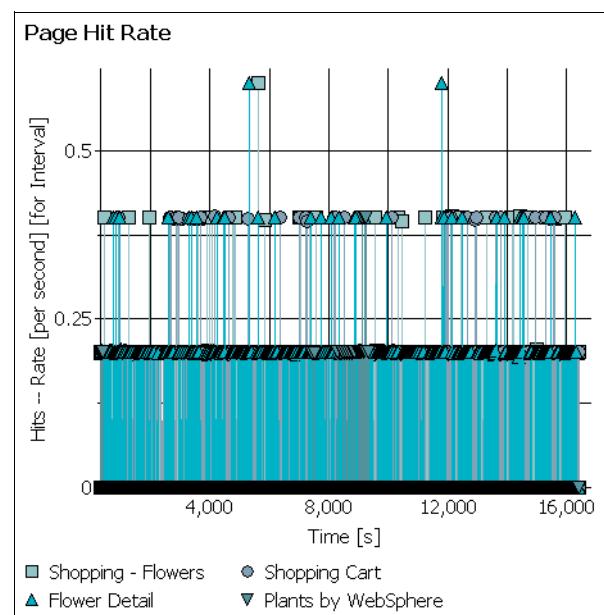


Figure 7-22 Page Throughput tab of the HTTP Performance Report

Although not always included on a default report, many protocols provide hit and attempt rates for the primary and even secondary targets. If desired, testers can create a custom report or report tab to analyze these rates in their *for Interval* or *for Run* format. Figure 7-23 shows the hit rates for individual pages in an HTTP test.

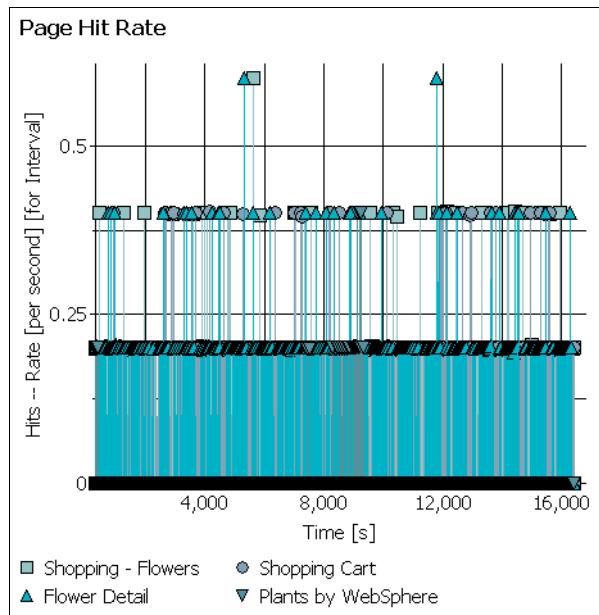


Figure 7-23 Custom report tab displaying hit rates for individual pages

As in all Performance Tester report graphics, if the amount of information presented on a graphic hinders analysis, filters can be used to focus on a particular page. Filtering is done through the context menu item, *Apply Filter*. Filters can also be applied in series and edited individually through the Performance Test Runs view, *Edit Filter* context menu, located on the filter's icon (Figure 7-24).

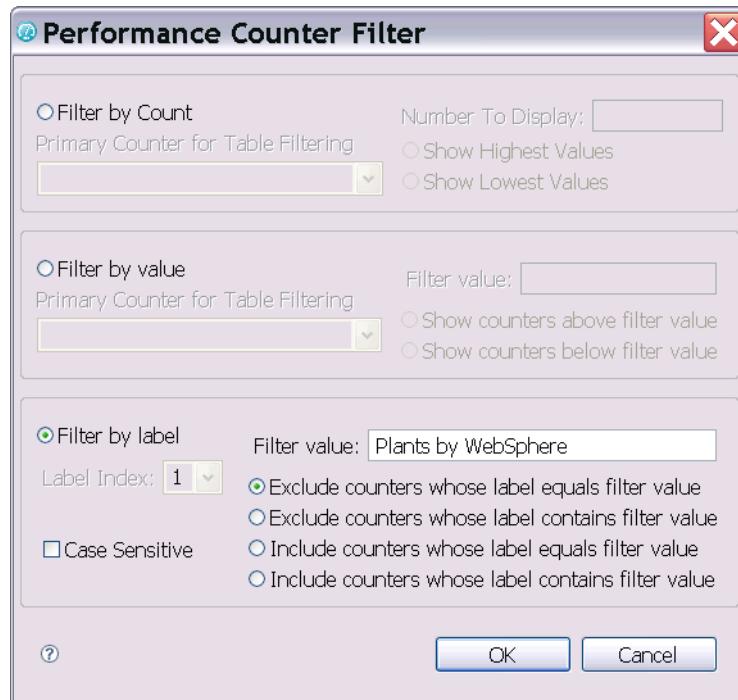


Figure 7-24 Editing an individual filter on a Performance Tester graphic

7.4 Using reports to find time consuming application steps

When testing is aimed at driving improvement, it is very useful to narrow in on the system components, both hardware and software, which are the system bottlenecks. To enable the tester to accomplish this, Performance Tester provides response time breakdown analysis (RTB). RTB allows drill-down for response analysis on various tiers of a multi-tier system. When problematic logic is discovered, the tester can navigate to the area in the system code-base that contains the problem. Performance Tester also does resource monitoring.

Resource monitoring allows the tester to capture resource usage on the various servers that make up the system under test. Correlation between poor response and resource over-utilization can easily be performed. Refer to Chapter 8, “Resource monitoring” on page 211 and Chapter 9, “Application monitoring” on page 255 to learn more about the systems supported for resource monitoring and response time breakdown and how to configure your system for each analysis.

In an example to follow, response time breakdown and resource monitoring will be used to find hardware and software bottlenecks on an IBM WebSphere Application Server based system.

7.4.1 Data collection configuration

To collect resource monitoring and response time breakdown data, the amount of each type of data to be collected must be specified. In addition, the collection method to use for resource monitoring data must be specified. This specification is done in the schedule editor.

Configuration of resource monitoring data collection

In the schedule editor with the schedule root selected, tabs appear in the details area of the editor (the right side of the editor). One of the tabs on this details area is *Resource Monitoring*. On the Resource Monitoring tab, data collection is configured by specifying location assets corresponding to the host machine of interest as shown in Figure 7-25.

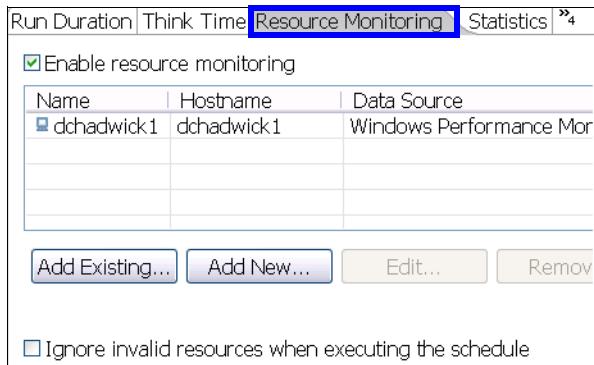


Figure 7-25 Resource Monitoring tab of the schedule editor

When *Add* is clicked and a location is selected, the tester is then presented with the Resource Monitoring wizard. The left side of the wizard contains three types of resource monitoring data sources. The data source options available are IBM Tivoli Monitoring, UNIX rstatd, and Windows Performance Monitor. When a source is selected, the user is prompted for authentication information if applicable (Figure 7-26).

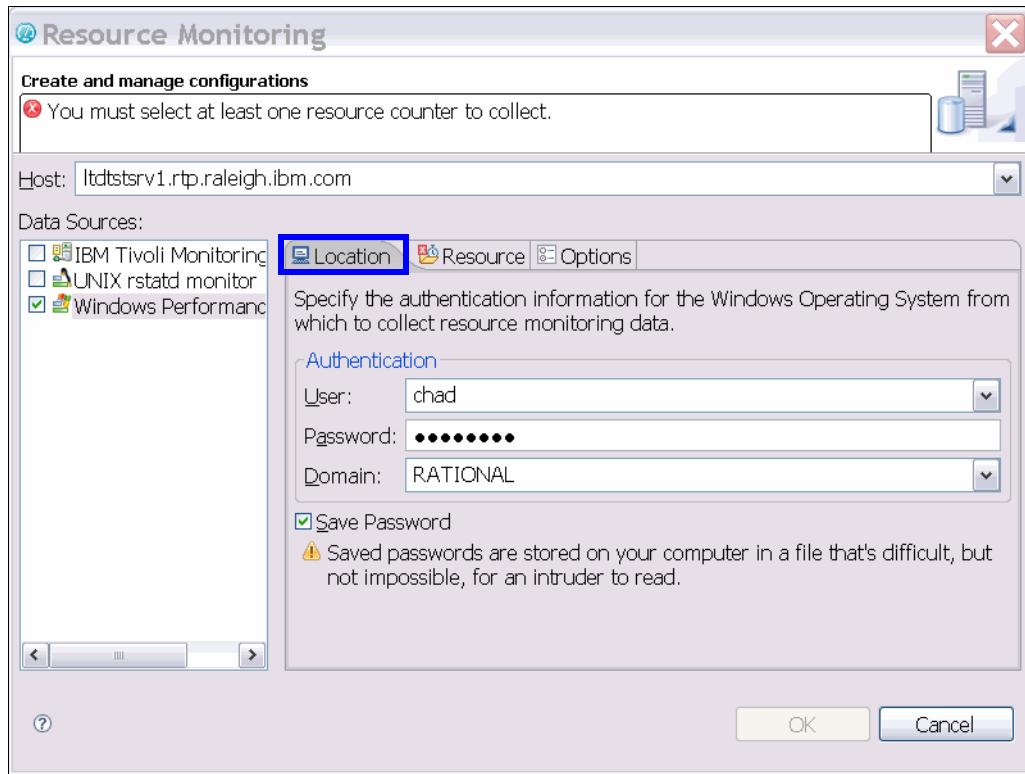


Figure 7-26 Resource Monitoring wizard Location tab

The 2nd tab of the wizard allows the user to select the counters that they wish to collect from all that the source has to offer. Upon first edit, a default set of counter is preselected. If the information that the tester wants to gather is not in the preselected list, clearing *Show only selected counters* causes a complete list of counters from the data source to be displayed.

We do not recommend that every counter from a data source be selected, because many are redundant and the amount of data collected can become great. Testers will find that there are many very useful counters available that enable them to perform a thorough analysis of the hardware utilization of their host. For this example, counters are selected to summarize memory, CPU, and network utilization on the host (Figure 7-27).

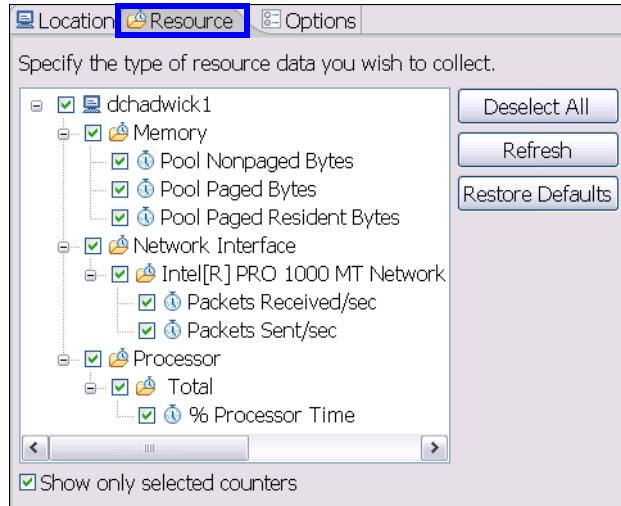


Figure 7-27 Resource Monitoring wizard Resource tab

The 3rd tab of the wizard allows the tester to specify the polling interval to use when collecting data from the sources. It generally is good practice to capture resource data on the same interval as all other Performance Tester statistics.

7.5 Some common scenarios for further analysis

Here are some common usage scenarios that lead to different analysis goals. In each of these scenarios, the test schedule is adjusted, changing the workload and data collection parameters to achieve the analysis desired.

Using reports for end-to-end transaction timing

Consider these recommendations:

- ▶ Use the default (performance) report to measure page response times.
- ▶ Check for time correlation of results.
- ▶ Check for normally distributed data (mean / std dev > 3).
- ▶ Analyze the performance within time ranges (steady state data only).
- ▶ Use percentile reports for HTTP page response times.
- ▶ Measure business transaction rates using transaction reports.

Configuration of response time breakdown data collection

Consider these recommendations:

- ▶ Run 50% full load test for 10-15 minute at steady state.
- ▶ Examine slow page responses/transactions → get to page element (request).
- ▶ Breakdown request to J2EE method calls causing slow response.
- ▶ Report method call data to development for tuning.
- ▶ Re-run to validate tuning.

Using reports to find maximum system throughput

Consider these recommendations:

- ▶ Run a ramp-up at 1 user/second to 150% of expected load.
- ▶ Observe hit rate versus time graph to watch for plateau.
- ▶ Verify proper operation up to and beyond full capacity hit rate.
- ▶ Obtain time where hit rates flatten out and read the number of active users.
- ▶ Check for system resource exhaustion to find what system has reached capacity.
- ▶ Run extended full capacity test at that load level to verify that it is sustainable.

Using reports to verify test agent fidelity of playback and measurement

Consider these recommendations:

- ▶ Turn on resource monitoring of test agent machine.
- ▶ Turn on informational logging of execution plug-in to check heap status.
- ▶ Adjust OS tuning for file handles, sockets, or >1G process size limits as needed.
- ▶ Run a ramp-up at 1 user/second to 1000 or 1500 virtual users on a single agent.
- ▶ Keep track of CPU utilization (<70%) to see if there are adequate resources.

Using reports to characterize full load system performance

Consider these recommendations:

- ▶ Turn on resource monitoring and turn off transaction breakdown.
- ▶ Turn logging on for 5 users per user group at primary level to get percentile data.
- ▶ Ramp up workload to 100% load point slowly enough to permit successful logins.
- ▶ Observe hit rates and response time stability for a time long enough to collect over 100 samples per response time and 20+ business transactions whenever possible.
- ▶ Terminate run with a two-minute grace period before a hard user shutdown.
- ▶ Verify test agents not overloaded using CPU monitoring on test agent.
- ▶ Verify proper transaction operation by running VP report.
- ▶ Run the transaction report to obtain transaction throughput rates after filtering for steady state range.
- ▶ Run percentile report for HTTP response times for steady state range.
- ▶ Export results from performance transaction percentile and VP reports.



Resource monitoring

This chapter describes the various resource monitoring data collectors available in Performance Tester. Here we discuss the architecture of each data collector and how each is configured when using Performance Tester.

We cover the following topics:

- ▶ Overview of resource monitoring
- ▶ Configuring a performance schedule
- ▶ Monitoring using Windows Performance Monitor
- ▶ Monitoring using rstatd for Linux/UNIX systems
- ▶ Monitoring using IBM Tivoli Monitoring
- ▶ Customizing reports and overlaying counters

8.1 Overview of resource monitoring

Resource monitoring is a term used to describe the observation of a property or asset over time. Typically, these observations are numerical facts or data, also known as statistical data. Most often, within the performance testing space, a resource can be, but is not limited to, a physical system or a process executing on a system. Resource monitoring is crucial when trying to determine the problem with an application that is failing to perform at reasonable levels, because this type of monitoring allows a performance tester to determine whether there is a lack of system-level resources or an issue with the application itself.

The resource monitoring feature in Rational Performance Tester allows real-time monitoring of systems and system processes. This granularity enables problem determination analysts to diagnose a problem accurately; because the resource monitoring data can help an analyst determine if the failure was caused by the lack of system resources (such as memory) or a particular process (such as a database) involved in an applications execution.

There are three resource monitoring data collectors provided in Rational Performance Tester:

- ▶ Windows Performance Monitor for Microsoft® Windows systems
- ▶ rstatd for Linux/UNIX® systems
- ▶ IBM Tivoli Monitoring (ITM) for monitoring a variety of platforms

Data collectors extend a generic Resource Monitoring platform, which is included in Performance Tester (Figure 8-1). Additional data collectors can be added into the product using Eclipse Platform extensions, which will then be automatically loaded by the Resource Monitoring platform.

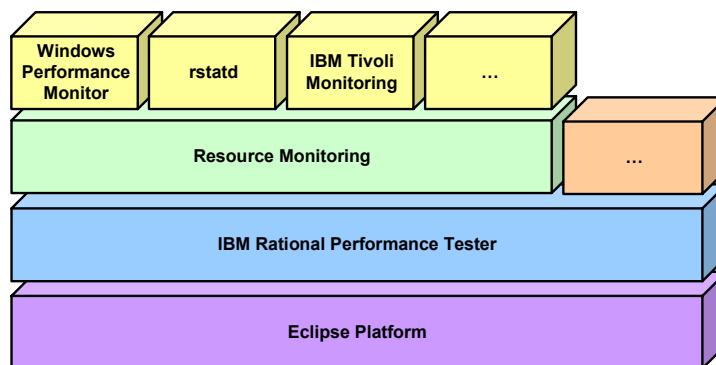


Figure 8-1 IBM Rational Performance Tester stack architecture

Because each data collector will have a specific mechanism for retrieving data from a remote host, the Resource Monitoring platform is designed to allow custom Java code to be written to implement the data collection mechanism using a common interface.

When the data is retrieved, the data must be transformed into the Eclipse Test & Performance Tools Platform (TPTP) statistical model format. This model format is based on the Eclipse Modeling Framework (EMF) and is designed to be a generic method of storing statistical data using XML. The open source Eclipse TPTP project provides APIs that abstract the details of storing this data and make it easy to store retrieved data.

All of the data collectors provided in Performance Tester have a specific mechanism of retrieving data. However, when the data has been gathered, the collectors transform this data into the common statistical model format. Thereby, Performance Tester and other Eclipse-based views are able to render reports based on this data.

8.2 Configuring a performance schedule

Resource monitoring is enabled when configuring a performance schedule. Selecting the top-most node in the Schedule Contents section of the Performance Schedule Editor (Figure 8-2) displays configuration parameters available in the Schedule Element Details section. A tab labelled *Resource Monitoring* contains the configuration parameters that are used when the schedule in context is executed.

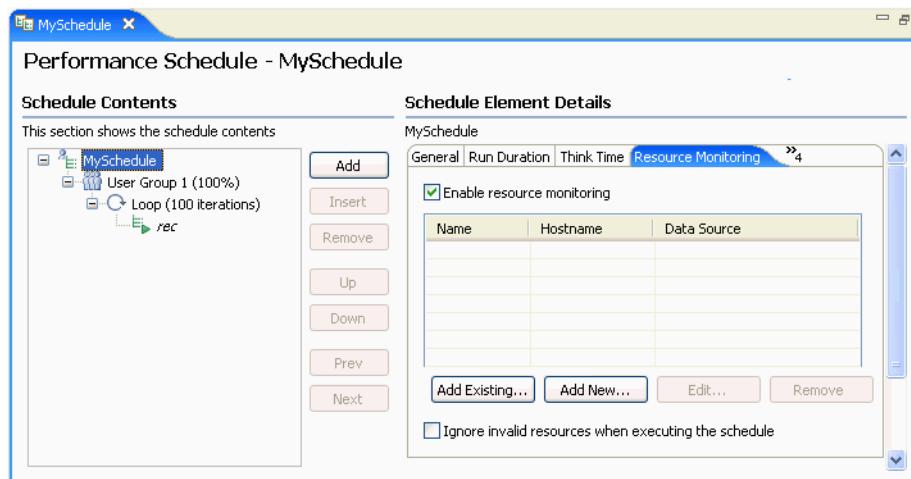


Figure 8-2 Performance Schedule Editor

Let us now explain some of the choices and actions:

- ▶ **Enabled resource monitoring:** This check box activates the resource monitoring feature for the performance schedule you are working with. It can be used as an on or off switch for resource monitoring as a whole.
- ▶ **Ignore invalid resources when executing the schedule:** Select this check box to suppress error messages about resource monitoring data sources. These errors can occur if the data sources are unreachable or invalid. If you select this option, you must view logs to see the error messages.
- ▶ The **Data Source** table in the Resource Monitoring tab displays the configured data sources that are collected from when this schedule executes. If this is a new schedule, the Data Source table will be empty. If you have previously added resource monitoring data sources to this schedule, you can *Edit* or *Remove* them. Remove does not delete the data source from the file system; it merely removes it from this view because other test schedules or applications might still use the data source.
- ▶ Clicking **Add New** (Figure 8-3) creates and allows a user to configure a new resource monitoring location:
 - **Host.** Indicates the target host name (Internet Protocol (IP) address for fully qualified host name) to monitor during the execution of the performance schedule.

Tips:

- ▶ When using IBM Tivoli Monitoring as the data source, the host field is the location of the resource management collection agent (or monitoring agent) and not the Tivoli Enterprise Monitoring Server.
- ▶ When using Windows Performance Monitor or UNIX rstatd monitor as the data source, the host field is the target Windows or UNIX system that has to be monitored.
- Numerous data collectors are available in the Data Sources table. If required, several data sources can be configured for one location by selecting the desired data sources from the check box list on the left-hand side of the dialog.

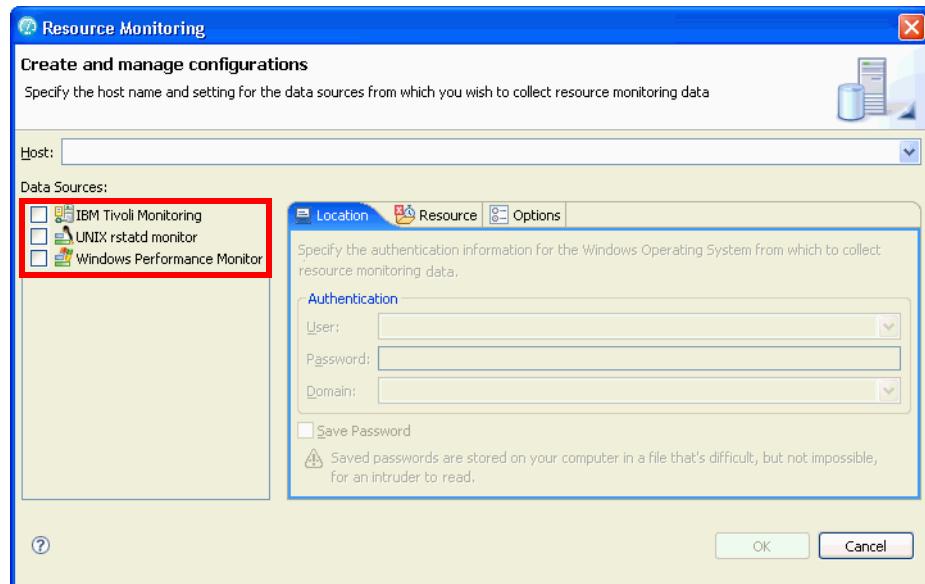


Figure 8-3 Create and configure a new resource monitoring location

- ▶ Click **Add Existing** (Figure 8-4) if you have existing locations in the workspace that you want to add and configure to the current working performance schedule.

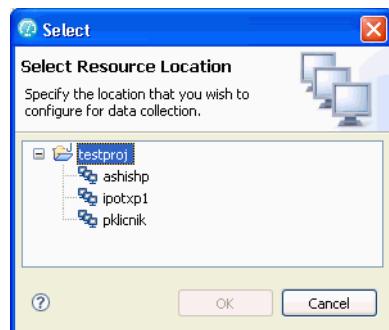


Figure 8-4 Add existing location to a performance schedule for resource monitoring

When you have enabled resource monitoring, you have to specify the data sources to collect from.

Any configuration changes you make for a particular data source are stored with that location. This means that you have to set up a data source only once. If you export a schedule, it will contain the data source configuration information. This includes potentially sensitive information, such as stored passwords.

It is important to note that to capture accurate resource monitoring data, you must ensure that the clocks on all systems are synchronized. If you do not synchronize the clocks on the workbench and on all of the systems under test, resource counters are displayed inaccurately (with respect to time) in the reports.

8.3 Monitoring using Windows Performance Monitor

The Microsoft Windows operating system contains an embedded measurement infrastructure that can be accessed either through the Windows registry or using application programming interfaces (APIs). The measurement infrastructure is a part of the Windows Management Infrastructure (WMI) and allows for real-time monitoring of approximately one thousand performance counters.

8.3.1 Performance counters

Performance counters indicate how well the operating system, an application, or a service is performing. Each counter is designated a specific metric to collect; for example, a counter named *% Processor Time* will collect the percentage of elapsed time that the processor spends to execute a non-idle thread and can be utilized as an indicator of processor activity. Capturing the measurements of such counters over time assists analysts to discover system bottlenecks and fine-tune a system or an application's performance.

Within the measurement infrastructure, counters are structured using a two-level hierarchy. The first level consists of the *performance object*, which is a unit that can model a physical device or abstract component in a system. The second level consists of the *performance counter*, which is an attribute of the performance object. Table 8-1 provides a list of three performance counters and the performance objects under which they are categorized.

Table 8-1 Windows Performance Monitor objects and counters

| Performance object | Performance counter |
|--------------------|--|
| Processor | % User Time % Processor Time Interrupts/sec |
| Memory | Page Reads/sec Page Writes/sec Available KBytes |
| Logical Disk | Disk Read Bytes/sec Disk Write Bytes/sec % Disk Time |

8.3.2 Data collection

The resource monitoring data displayed by Performance Tester when using Windows Performance Monitor is collected by leveraging the existing measurement infrastructure already present in the Microsoft Windows operating system. Therefore, this data collector is considered to be *agent-less*, because a user of Performance Tester does not have to install any additional software to the operating system that is being monitored.

Performance counter data is stored within the Windows Registry and can be programmatically accessed using two methods (Figure 8-5):

- ▶ The first method is to access the Windows Registry directly, however, this approach can be complex and cumbersome.
- ▶ The second method, which is the recommended method, is to use the performance data helper (PDH) API. The PDH interface is an abstraction of the details required to retrieve performance object and performance counter data from the Windows registry. In addition, the interface automatically returns values that are adjusted for appropriate units and scale. The interface is also the foundation for the Windows Performance Monitor (also known as *Perfmon*).

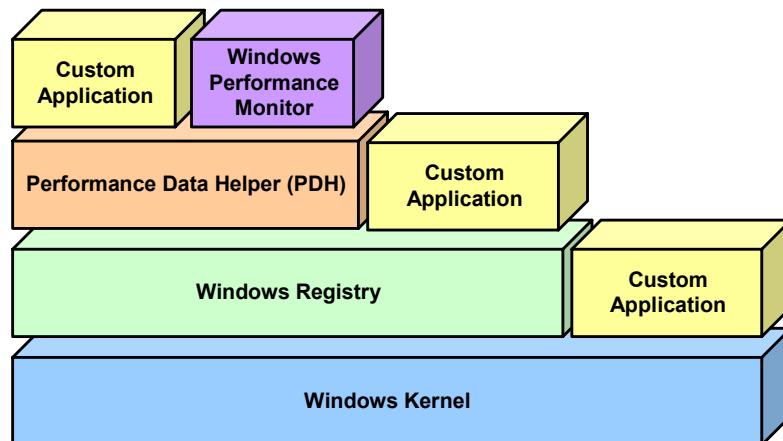


Figure 8-5 Microsoft Windows Performance Monitor stack architecture

Physical devices (such as processors, memory, and network cards) and applications (such as operating systems, databases, servers, and third-party drivers) can have special counters built into them that are monitored by the Windows Kernel. This approach allows for extensibility of the standard set of counters monitored by the operating system alone.

Moreover, any applications already using the Windows registry or the PDH interface automatically are able to monitor these special counters, when the counters have been registered for collection.

In the case of Performance Tester, a custom application (pdh.dll) was built using the PDH interface to enable data collection from systems running Microsoft Windows. The custom application integrates with the Eclipse Platform in the form of a plug-in extension to Performance Tester. This plug-in interacts with the custom application using the Java Native Interface (JNI™).

8.3.3 System configuration

In this section we describe how to configure the system for the Windows Performance Monitor.

Performance detail level

The measurement infrastructure contains approximately one thousand performance counters, and the quantity is increasing with newer releases of Microsoft Windows. Presenting a user with a large volume of counters poses a usability issue, and therefore, there is a configuration parameter that should be used with the PDH interface to help limit the number of performance counters presented to the user.

The following is a list of the different performance detail levels available from the PDH interface.

- ▶ **Novice**—Indicates that this counter might be meaningful to most users. This is the most common counter detail level.
- ▶ **Advanced**—Indicates that this counter is likely to be useful only to advanced users.
- ▶ **Expert**—Indicates that this counter is likely to be useful only to the most advanced users.
- ▶ **Wizard**—Indicates that this counter is not likely to be useful to any users.

In the case of Performance Tester, the custom application (pdh.dll) uses the *Advanced* detail level. This detail level includes generic counters (including processor, disk, memory, and more) and specific counters (including network card interfaces). In this release, this detail level is not configurable.

Network interface

A user of the Windows Performance Monitor data collector must ensure that the network interface on the physical machine that data is being collected from is configured correctly. Users must ensure that the connected network interface has *File and Printer Sharing for Microsoft Networks* enabled (Figure 8-6).

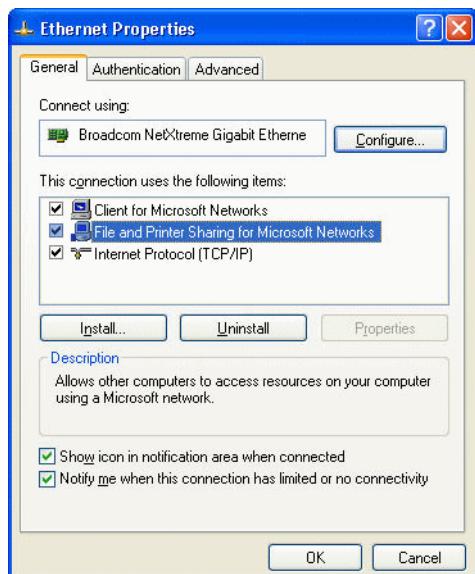


Figure 8-6 Ethernet network connection properties in Microsoft Windows

This is required because the PDH uses the Windows `net use` command to establish a connection to remote machines for data collection. This service can be enabled by opening the Properties dialog on a network connection listed in the Control Panel of any Microsoft Windows system.

A user of Performance Tester can collect performance counter data from any system running the Microsoft Windows operating system, only when the Performance Tester workbench client is executed on a Microsoft Windows operating system. This limitation is present because the PDH uses the operating systems at the client and target endpoint for communication.

8.3.4 Configuring a performance schedule

To configure resource monitoring using Windows Performance Monitor, create a new location, or add an existing location to a given performance schedule.

Enter a target Windows host name from which to collect data from and select *Windows Performance Monitor* from the list of available data sources (Figure 8-3 on page 215).

Note: The host you want to monitor must be accessible through the Windows network. Typically, if you are able to connect to a shared hard disk drive on the remote host from the workbench, then you will also be able to use the remote host as a source of resource monitoring data. If the remote host is not accessible through the Windows network, an error message will display when you attempt to use the Resource page to select the type of data that you want to capture.

On the **Location** tab, type the user name, password, and domain (Figure 8-7).

- ▶ The user name and password must match to a Windows user account on the target host system.
- ▶ The domain is optional, only required if you need to perform cross-domain authentication.
- ▶ Select *Save Password* to save your password locally. If you do not save your password, you might be prompted for it (depending on the host system configuration) when editing the configured location or when running performance schedules that use the location.

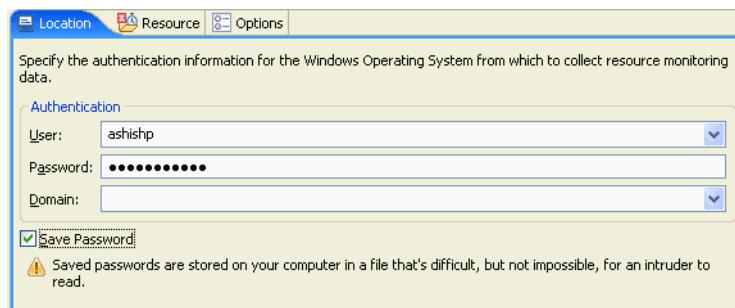


Figure 8-7 Windows Performance Monitor Location configuration

On the **Resource** tab, select the type of data that you want to capture (Figure 8-8). The tree view shows the host and all of its respective counter groups and counters. The tree is organized by performance objects, denoted by folder icons, and performance counters, denoted by stop-watch icons.

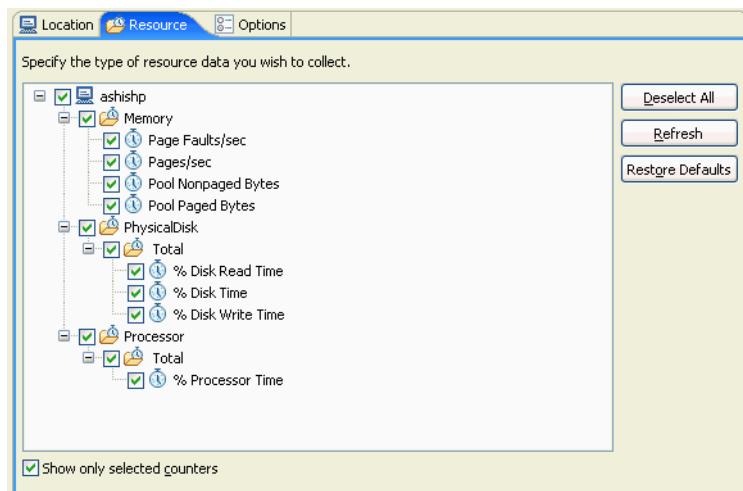


Figure 8-8 Windows Performance Monitor Resource tab

Clearing *Show only selected counters* allows you to see all available counters (Figure 8-9). Be selective; monitoring all possible resource data requires substantial amounts of memory. Hover over a counter with your mouse to see details about what that counter measures.

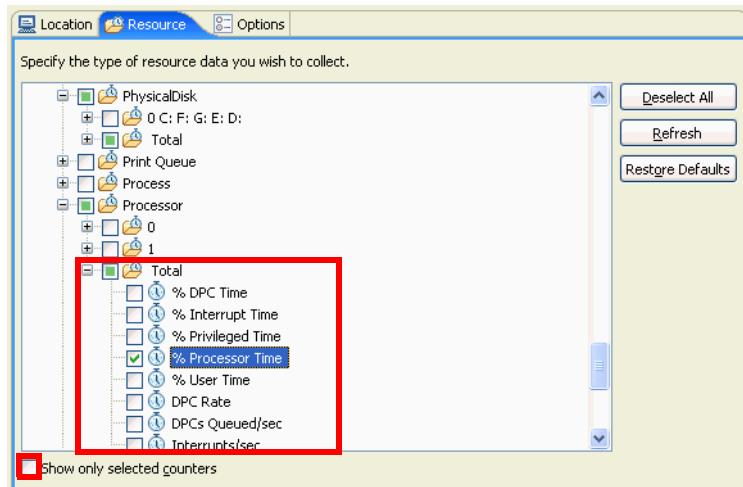


Figure 8-9 Windows Performance Monitor Resource tab showing all available counters

On the **Options** tab, the time interval properties can be configured (Figure 8-10):

- ▶ Type the Polling Interval in seconds, for collecting resource data. For example, if you accept the default of five seconds, counter information will be collected at five second intervals from the specified host during the schedule run.
- ▶ Type the Timeout Interval in seconds. If the resource monitoring host does not respond within this amount of time during a schedule run, an error is logged.

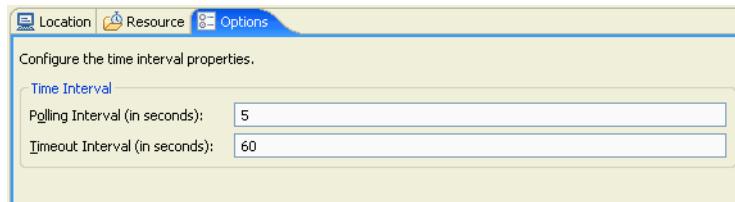


Figure 8-10 Windows Performance Monitor Options tab

When the configuration is complete and saved, the resource monitoring location is added to the performance schedule (Figure 8-11).

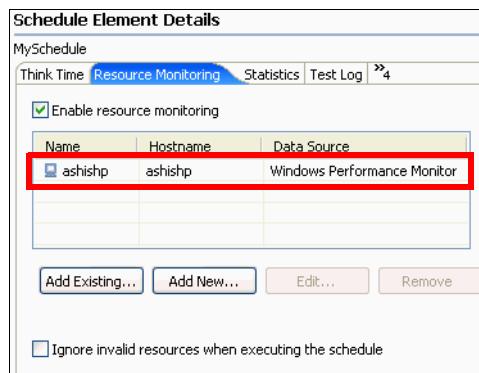


Figure 8-11 Resource Monitoring configured using Windows Performance Monitor

8.4 Monitoring using *rstatd* for Linux/UNIX systems

The rstat daemon (herein referred to as *rstatd* and sometimes referred to as *rpc.rstatd*) is a Remote Procedure Call (RPC) server that returns performance statistics for a UNIX-based operating system. The performance statistics are retrieved from the kernel itself.

This daemon might already be installed and running on most Solaris™ and Linux installations, however, this configuration will depend on the vendor and distribution. The rstat daemon is normally started by the inetd daemon.

8.4.1 Performance counters

The statistics monitored are restricted to fifteen performance counters, which include processor, disk, memory, and network utilization. The available counters are shown in Table 8-2.

Table 8-2 Performance counters for the rstat daemon on Linux/UNIX

| Performance counter | Description |
|---|---|
| Percentage User CPU Time | Percentage of non-idle processor time spent in user mode. |
| Percentage IOWAIT CPU Time | Percentage of non-idle processor time spent waiting for I/O to complete. |
| Percentage of System CPU Time | Percentage of non-idle processor time spent in system mode. |
| Percentage of Idle CPU Time | Percentage of time that the processor is idle. |
| Total Disk Transfers per second | Total number of disk transfers on each of the disk interfaces (per second). |
| Total VM Pages Paged IN per second | Total number of pages paged in per second (paging versus swapping: ^{a)}) |
| Total VM Pages Paged OUT per second | Total number of pages paged out per second (paging versus swapping: ^{a)}) |
| Total VM Pages Swapped IN per second | Total number of pages swapped in per second (paging versus swapping: ^{a)}) |
| Total VM Pages Swapped OUT per second | Total number of pages swapped out per second (paging versus swapping: ^{a)}) |
| Total Interrupts per second | Total number of interrupts per second. |
| Total Inbound packets on all interfaces per second | Total number of inbound packets on all interfaces per second |
| Total Inbound errors on all interfaces per second | Total number of inbound errors on all interfaces per second |
| Total Outbound packets on all interfaces per second | Total number of outbound packets on all interfaces per second |

| Performance counter | Description |
|---|---|
| Total Outbound errors on all interfaces per second | Total number of outbound errors on all interfaces per second |
| Total collisions seen on all interfaces per second | Total number of collisions seen on all interfaces per second |
| Total context switches per second | Total number of context switches per second |
| Average number of jobs in run queue (1 minute average) | Number of jobs in the run queue averaged over the last 1 minute. ^b |
| Average number of jobs in run queue (5 minute average) | Number of jobs in the run queue averaged over the last 5 minutes. ^b |
| Average number of jobs in run queue (15 minute average) | Number of jobs in the run queue averaged over the last 15 minutes. ^b |

a. Paging is a light-weight mechanism whereby the operating system reads/writes pages of memory from/to the disk; swapping is similar to paging, but is more heavy-weight in that entire processes can have all their pages read/written from/to disk.

b. The jobs on the run queue are the processes that are ready to run on the processor.

8.4.2 Data collection

The rstat daemon is a native application that specifically operates on UNIX-based systems, such as Solaris, HP-UX, AIX, and Linux. Performance Tester integrates with the daemon through a Java-based RPC client, which is responsible for marshalling transactions to and processing the results from a rstat daemon. The RPC client is bundled within an Eclipse plug-in that extends the Resource Monitoring platform within Performance Tester.

When performance counters are selected to be monitored, a Remote Service Monitor (RSM™) program polls the counters at an interval and sends alerts to the client upon detection. The RSM program sends resource data to the client at reporting intervals set by the client or system administrator.

Depending on the setup of your account and the UNIX-based operating system, the system administrator might control the location and maintenance of the RSM programs that monitor your account. The RSM program obtains the performance data from the rstatd server, which returns performance statistics obtained from the kernel.

The version of rstatd that is supported in Performance Tester is the RSTATVERS_TIME version, or version 3. The rstatd is backwards compatible, therefore, versions greater than 3 will support the RSTATVERS_TIME edition. Historically, this version has been in existence well before 1995.

8.4.3 System configuration

In this section we describe how to configure the system for the rstatd daemon.

Installing rstatd on Linux

The rstatd daemon is freely available on the Internet for the Linux platform, and is already bundled in most UNIX-based platforms. Follow these installation instructions for rstatd on Linux using the open source version of rstatd available on SourceForge:

- ▶ Go to the Rstatd 4 Linux site and download the latest version:
<http://sourceforge.net/projects/rstated/>
- ▶ Extract the archive downloaded from the previous step, and follow the instructions in the install file. This file will be available after extracting the archive. Installation will require tools such as **make** and **gcc**, which are usually installed by default with most Linux distributions.
- ▶ The next step requires verifying that the rstatd daemon gets started through **inetd** daemon (or **xinetd** on some Linux distributions). Using a terminal, change directories to /etc/xinetd.d and create a file called rstatd, with the following contents:

```
service rstatd {  
    type = RPC  
    socket_type = dgram  
    protocol = udp  
    server = /usr/local/sbin/rpc.rstatd  
    wait = yes  
    user = root  
    rpc_version = 2-4  
    disable = no  
}
```

Ensure that the value specified by the *server* parameter is actually where the *rpc.rstatd* daemon is installed. If it is not installed in */usr/local/sbin* change the aforementioned parameters to reflect this configuration.

- ▶ Restart the *inetd* (or *xinetd*) daemon:

```
/etc/rc.d/init.d/xinetd restart
```

- ▶ Now rstatd should be running. Running the following command will help verify that rstatd is actually started:

```
rpcinfo -p localhost
```

The daemon should be listed several times, which is due to the different RPC versions available for clients to use.

Portmapper service

The rstatd uses the Remote Procedure Call (RPC) system to facilitate connecting to the system **portmapper** daemon running on a well-known port. The rstatd will use RPC to create a listener that asks the portmapper for a port to use for communication with a remote client. The portmapper selects an unused port and assigns it to the listener.

Performance Tester requires the portmapper daemon to be running on the remote Linux/UNIX system. Running the following command will help verify that the daemon is actually started:

```
rpcinfo -p localhost
```

This command will generate output similar to the following output:

```
program vers proto  port
    100000    2   tcp   111  portmapper
    100000    2   udp   111  portmapper
    100024    1   udp  32768  status
    100024    1   tcp  32769  status
    100001    2   udp  32770  rstatd
    100001    3   udp  32770  rstatd
    100001    4   udp  32770  rstatd
```

The output indicates that the portmapper is operating using the TCP and UDP protocols using port 111, the default port.

If the portmapper daemon is not listed in the output of the rpcinfo command, as shown in our example, the daemon might not be installed or not be configured to run. We recommend that you contact your system administrator and request that the portmapper RPC program be started.

To view the most commonly used RPC program numbers is held in a file called `/etc/.rpc` on each system. The file consists of a series of entries, one per service, such as:

```
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
```

| | | |
|---------------|--------|-------------------------|
| mountd | 100005 | mount showmount |
| ypbind | 100007 | |
| walld | 100008 | rwall shutdown |
| yppasswdd | 100009 | yppasswd |
| etherstatd | 100010 | etherstat |
| rquotad | 100011 | rquotaproq quota rquota |
| sprayd | 100012 | spray |
| 3270_mapper | 100013 | |
| rje_mapper | 100014 | |
| selection_svc | 100015 | selnsvc |
| database_svc | 100016 | |

8.4.4 Configuring a performance schedule

To configure resource monitoring using UNIX rstatd monitor, create a new location, or add an existing location to a given performance schedule.

Enter a target Linux/UNIX host name from which to collect data from and select the *UNIX rstatd monitor* item from the list of available data sources (Figure 8-3 on page 215).

On the *Resource* page, select the type of data you want to capture (Figure 8-12). The tree view shows all available performance counters, with a default set of counters pre-selected.

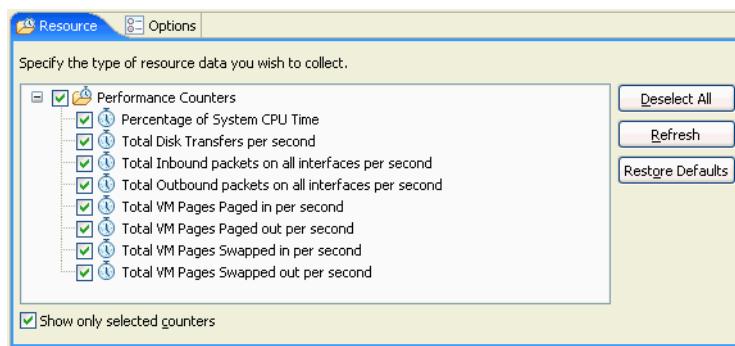


Figure 8-12 UNIX rstatd monitor Resource tab

Clearing *Show only selected counters* allows you to see all available counters (Figure 8-13). Be selective; monitoring all possible resource data requires substantial amounts of memory. Hover over a counter with your mouse to see details about what that counter measures.

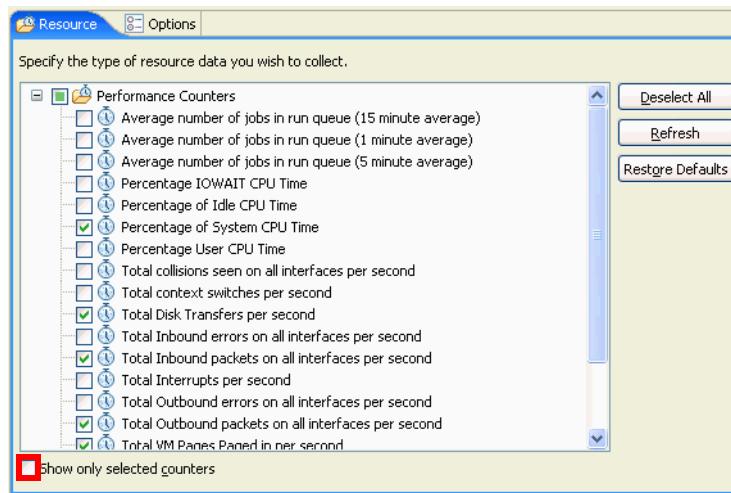


Figure 8-13 UNIX rstatd monitor Resource tab showing all available counters

On the *Options* tab, the time interval properties can be configured (Figure 8-14).

- ▶ Change the Connection information if needed. Typically, you only need to change this information if a firewall is blocking the default port. If the portmapper has been configured to use a different Protocol or Portmapper™ Port number, this area allows for custom configuration to be applied for the specific target Linux/UNIX host being monitored.
- ▶ Type the Polling Interval in seconds, for collecting resource data. For example, if you accept the default of 5 seconds, counter information will be collected at 5-second intervals from the specified host during the schedule run.
- ▶ Type the Timeout Interval in seconds. If the resource monitoring host does not respond within this amount of time during a schedule run, an error is logged.

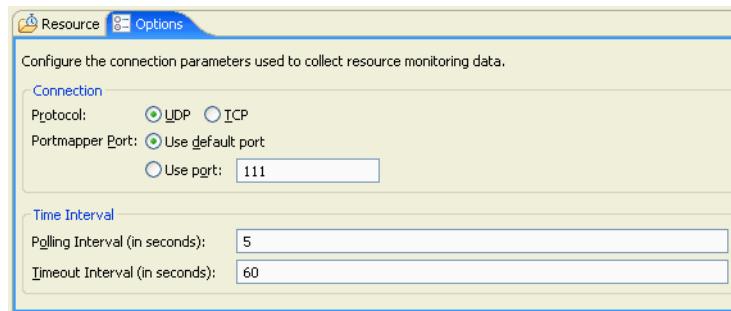


Figure 8-14 UNIX rstatd monitor Options tab

When the configuration is complete and saved, the resource monitoring location is added to the performance schedule (Figure 8-15).

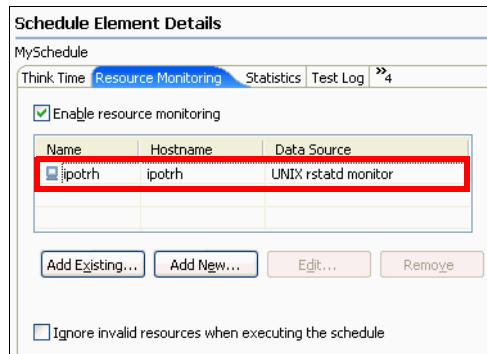


Figure 8-15 Resource Monitoring configured using UNIX rstatd monitor

8.5 Monitoring using IBM Tivoli Monitoring

IBM Tivoli Monitoring (ITM) monitors and manages system and network applications on a variety of platforms and keeps track of the availability and performance of all parts of your enterprise. IBM Tivoli Monitoring provides reports you can use to track trends and troubleshoot problems.

8.5.1 Components of IBM Tivoli Monitoring

The system architecture of IBM Tivoli Monitoring is shown in Figure 8-16.

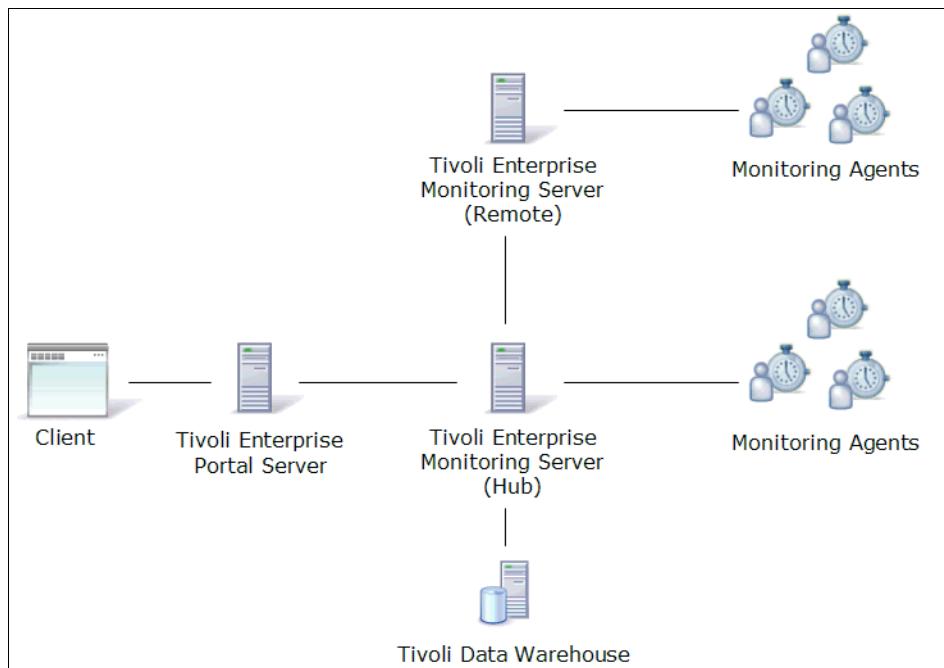


Figure 8-16 IBM Tivoli Monitoring system architecture

IBM Tivoli Monitoring consists of the following components:

- ▶ A **Tivoli Enterprise Monitoring Server** (referred to as the monitoring server), which acts as a collection and control point for alerts received from the agents, and collects their performance and availability data. There are two types of monitoring servers: hub and remote.
- ▶ A **Tivoli Enterprise Portal Server** (referred to as the portal server). Placed between the client and the monitoring server, this enables retrieval, manipulation, and analysis of data from the agents.
- ▶ A **Tivoli Enterprise Portal Client**, with a Java-based user interface for viewing and monitoring your enterprise. Tivoli Enterprise Portal offers two modes of operation: desktop and browser.
- ▶ **Monitoring Agents** installed on the systems or subsystems you want to monitor. These agents collect and distribute data to a monitoring server.
- ▶ **Tivoli Data Warehouse** for storing historical data collected from agents in your environment. The data warehouse is located on a DB2®, Oracle, or Microsoft SQL database. To collect information to store in this database, you must install the Warehouse Proxy agent. To perform aggregation and pruning functions on the data, install the Warehouse Summarization and Pruning agent.

- ▶ **Tivoli Enterprise Console®** event synchronization component (not shown) for synchronizing the status of situation events that are forwarded to the event server. When the status of an event is updated because of Tivoli Enterprise Console rules or operator actions, the update is sent to the monitoring server, and the updated status is reflected in both the Situation Event Console and the Tivoli Enterprise Console event viewer.

8.5.2 Monitoring agents

Today, there are over one hundred ITM monitoring agents available from IBM and third-party software vendors. Each agent is responsible for data collection from systems, subsystems, or applications. The collected data is passed from the monitoring agents to the portal server through the monitoring server. An agent interacts with a single system or application and, in most cases, resides on the same computer where the system or application is running. There are two types of monitoring agents:

- ▶ Operating system (OS) agents that monitor the availability and performance of the computers in your monitoring environment. An example of an OS agent is the Monitoring Agent for Windows OS, which monitors Windows XP, Windows 2000, and Windows 2003 operating systems.
- ▶ Other agents (referred to as application agents or non-OS agents) that monitor the availability and performance of systems, subsystems, and applications. An example of a non-OS agent is IBM Tivoli Monitoring for Microsoft Exchange, which monitors the Microsoft Exchange Server.

IBM Tivoli Monitoring also provides a customized agent called the *Universal Agent*. The Universal Agent can monitor any data that is collectable in a given environment. For example, the Universal Agent can monitor the status of your company Web site to ensure that it is available.

Agents run on z/OS®, UNIX, Windows XP Professional Edition, Windows 2000, Windows 2003 Server, HP-UX; however, not all agents are supported on all platforms.

Each agent has special metadata that describes itself. This metadata is required when integrating data collection into a custom client, such as Performance Tester. Today, Performance Tester is capable of data collection from the following ITM Monitoring Agents:

- ▶ Operating system agents:
 - Monitoring Agent for Windows OS
 - Monitoring Agent for Linux OS
 - Monitoring Agent for UNIX OS
 - Monitoring Agent for z/OS

- ▶ Application agents:
 - Monitoring Agent for Citrix
 - Monitoring Agent for IBM DB2
 - Monitoring Agent for IBM WebSphere Application Server
 - Monitoring Agent for IBM WebSphere MQ
 - Monitoring Agent for Oracle Database
 - Monitoring Agent for SNMP-MIB2 (only)
 - Monitoring Agent for IBM Tivoli Composite Application Manager for WebSphere

8.5.3 Data collection

Retrieving data from ITM is accomplished by making SOAP requests to the embedded SOAP server contained within the Tivoli Enterprise Monitoring Server. It is important to note that in the IBM Tivoli Monitoring documentation, the SOAP server is often referred to as a *Web service*, even though there is no Web Service Definition Language (WSDL) publicly published.

When data is retrieved using SOAP requests, the well formed XML responses are transformed into the Eclipse Test and Performance Tools Platform (TPTP) statistical model format. This model format is based on the Eclipse Modeling Framework (EMF) and is designed as a generic method of storing and representing statistical data, despite which system or data collector the data points are observed from. The generic model format allows Performance Tester and other Eclipse-based views to render reports based on this data.

As each event (a statistical data point) is observed, it is transformed into a TPTP Statistical model event on-the-fly. This event is then loaded into the TPTP Data Loader, which persists the event and triggers any registered user interfaces to update based on this new event.

The integration between Performance Tester and IBM Tivoli Monitoring is shown in Figure 8-17.

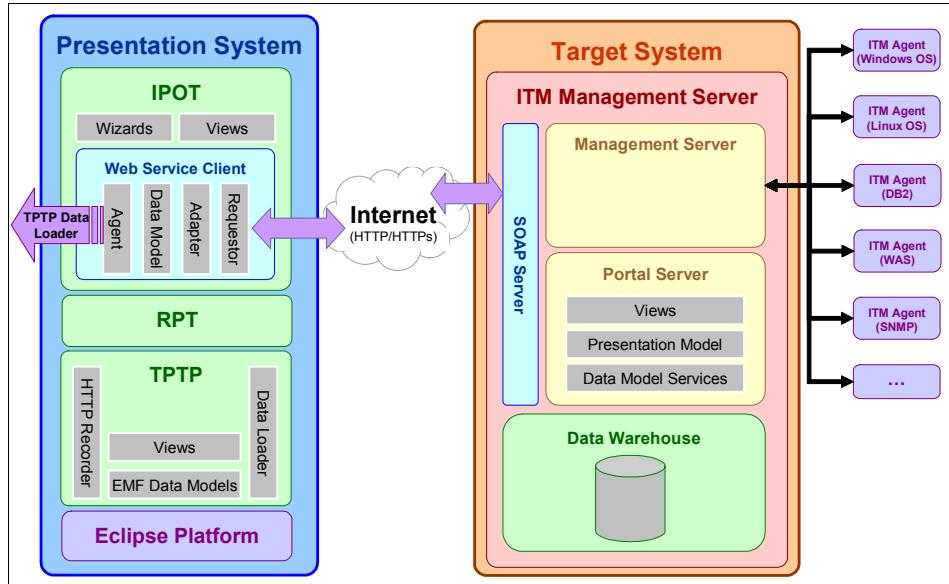


Figure 8-17 Rational Performance Tester integration with IBM Tivoli Monitoring

There are two mechanisms for statistical data that can be collected from ITM:

- ▶ **Real-time:** Real-time data collection requires making SOAP requests to the monitoring server requesting for the most recently acquired data point.
- ▶ **Importing historically:** Importing historical statistical data is possible because ITM is a managed solution, where the data observed over time is persisted to the Tivoli Data Warehouse.

A SOAP request to the monitoring server consists of:

- ▶ Constructing an XML fragment that corresponds to one of the SOAP methods described in the ITM documentation. Refer to the *Administering OMEGAMON® Products* document located at:
`http://publib.boulder.ibm.com/tivid2/td/IBMTivoliCandleNetPortall.9.51nk.html`
- ▶ Sending the XML fragment as the entity-body of an HTTP POST request to the URL:
`http://<MONITORING_SERVER>:1920//cms/soap/kshhsoap.htm`
 - <MONITORING_SERVER> refers to the name of the host on which the monitoring server process is running.
 - The three slashes (///) in the URL are required by the monitoring server.

- ▶ Setting the following properties in the HTTP header:
 - interfacename
 - CT_SOAP
 - messagetype: Call

Here is a sample HTTP POST request that can be used to query ITM for some data.

```
POST http://localhost:1920///kdsmain/soap HTTP/1.0
Accept: /*
Accept-Language: en-us
Referer: http://localhost:1920///kdsmain/soap/kshhsoap.htm
interfacename: CT_SOAP
messagetype: Call
Content-Type: text/xml
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
            .NET CLR 1.1.4322;.NET CLR 1.0.3705)
Host: localhost:1920
Content-Length: 96
Pragma: no-cache
```

```
<CT_Get><userid>sysadmin</userid><password></password>
    <object>Web_Applications</object></CT_Get>
```

- CT_Get is a command that the ITM SOAP server understands and declares the opening of a new transaction query.
- userid and password are used for authenticating with the SOAP server.
- object refers to the ITM object name for which we are querying (explained in more detail in the next section). In order to get the available agents, the ITM Object name is set to ManagedSystem.

Types of queries

In this section we list three types of queries.

Query monitor agents

Querying for monitoring agents that are available to the indicated monitoring server and collecting data:

- ▶ Example query:

```
<CT_Get><userid>sysadmin</userid><password></password>
    <object>ManagedSystem</object>
</CT_Get>
```

► Example response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-CHK:Success
    xmlns:SOAP-CHK = "http://soaptst1/soaptst/"
    xmlns="urn:candle-soap:attributes">
<TABLE name="04SRV.INODESTS">
<OBJECT>ManagedSystem</OBJECT>
<DATA>
    <ROW>
        <Timestamp>1050620105552004</Timestamp>
        <Name>RDANEK:UA</Name>
        <Managing_System>HUB_RDANEK</Managing_System>
        <ORIGINNODE>RDANEK:UA</ORIGINNODE>
        <Reason>FA</Reason>
        <Status>*OFFLINE</Status>
        <Product>UM</Product>
        <Version>04.10.00</Version>
        <Type>V</Type>
        <HLOCFLAG>L</HLOCFLAG>
        <HHOSTINFO>WinXP~5.1-SP1</HHOSTINFO>
        <HHOSTLOC></HHOSTLOC>
        <CT_Host_Address>ip:#169.254.198.66[38213]&lt;NM&gt;RDANEK&lt;/NM&gt;;
        </CT_Host_Address>
    </ROW>
</DATA>
</TABLE>
</SOAP-CHK:Success></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

The response is returned in a well-formed XML document structure. Now let us dissect the response:

- The OBJECT element echoes the same ITM object name that was sent in the initial SOAP.
- The DATA contains the result data of the query, which is of most interest to us. Each separate XML fragment of data is enclosed in a ROW element.
- Within each ROW element are the attributes which are known about for the given object. For example, Timestamp, Name, and Managing_System are some of the available attributes.
- The NAME element contains the name of the monitoring agent.
- The CT_Host_Address contains the host and/or address on which the Monitoring Agent is running.

Query recent data

Querying for recent data allows clients to retrieve the most recently acquired statistical data point by the monitoring server for a monitoring agent. This query would be used in the live monitoring scenario where periodically the statistical model is updated with new data points:

- ▶ Example query:

```
<CT_Get><userid>sysadmin</userid><password></password>
  <target>Primary:RDANEK:NT</target>
  <object>NT_Process</object>
  <attribute>Thread_Count</attribute>
  <afilter>ID_Process;EQ;988</afilter>
</CT_Get>
```

- A TARGET is the name of the monitoring agent as returned in the NAME element when querying for the available agents on a monitoring server.
- An OBJECT indicates the ITM Object that is being queried for data. This element is similar to a performance object in the Windows Management Infrastructure (WMI).
- (optional) ATTRIBUTE elements are the Performance Counters belonging to the ITM object (or performance object). For example, the NT_Process object has attributes named Process Name, % Processor time, % User Time. By specifying one or more attributes, one limits the results returned to the specified attributes.
- Filters, indicated by the AFILTER element, allow the query to specify the values of attributes for which the query should return data for. For example, when querying the NT_Process object, one might only want data for a certain process.

This example query returns the value of thread count for the process with an identifier of 988.

- ▶ Example response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-CHK:Success xmlns:SOAP-CHK = "http://soaptst1/soaptst/"
  xmlns="urn:candle-soap:attributes">
<TABLE name="KNT.WTPROCESS">
  <OBJECT>NT_Process</OBJECT>
    <DATA>
      <ROW>
        <Thread_Count dt="number">24</Thread_Count>
      </ROW>
    </DATA>
  </TABLE>
</SOAP-CHK:Success>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```

        </DATA>
    </TABLE>
</SOAP-CHK:Success></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

The response is returned in a well-formed XML document structure. Now let us dissect the response:

- The OBJECT element echoes the same ITM Object name that was sent in the initial SOAP.
- The DATA contains the result data of the query, which is of most interest to us. Each separate XML fragment of data is enclosed in a ROW element. In the example response, the value of the Thread count for the NT_Process object matching the process with an identifier of 988 is 24.

Query historical data

Querying for historical data allows the retrieval of statistical observations collected in the past for a monitoring agent. In order for this type of data collection to be enabled, historical data collection must have been configured for the desired monitoring agent through the monitoring server. The queries are identical to the preceding section except that we specify an additional tag in the query: <history>Y</history>:

► Example query:

```

<CT_Get><userid>sysadmin</userid><password></password>
    <target>Primary:RDANEK:NT</target>
    <object>NT_Process</object>
    <attribute>Timestamp</attribute>
    <attribute>Thread_Count</attribute>
    <history>Y</history>
    <afilter>Timestamp;GE;1050620132400000</afilter>
</CT_Get>
```

This query returns the thread count for all observations after June 20th, 2005 at 1:24 PM. A filter is used to specify a timestamp in order to only retrieve the historical data for the time period that is of interest. Querying for historical data without a timestamp filter results in the SOAP server returning all historical data, and this can overwhelm the system resources.

The timestamp is of the format CYYMMDDHHMMSSmmm:

- C is the century guard
- YY is the year
- First MM is the month
- DD is the day
- HH is the hour
- Second MM is the minute
- SS is the seconds
- mmm is the milliseconds

- ▶ Example response:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-CHK:Success xmlns:SOAP-CHK = "http://soapttest1/soapttest/">
    xmlns="urn:candle-soap:attributes">
        <TABLE name="KNT.WTPROCESS">
            <OBJECT>NT_Process</OBJECT>
            <DATA>
                <ROW>
                    <Timestamp>1050620132453887</Timestamp>
                    <Thread_Count dt="number">1</Thread_Count>
                </ROW>
                <ROW>
                    <Timestamp>1050620132453887</Timestamp>
                    <Thread_Count dt="number">64</Thread_Count>
                </ROW>
                <ROW>
                    <Timestamp>1050620132453887</Timestamp>
                    <Thread_Count dt="number">3</Thread_Count>
                </ROW>
            </DATA>
        </TABLE>
    </SOAP-CHK:Success></SOAP-ENV:Body></SOAP-ENV:Envelope>

```

8.5.4 Configuring a performance schedule

To configure resource monitoring using IBM Tivoli Monitoring, create a new location, or add an existing location to a given performance schedule.

Enter a target host name, where the ITM Monitoring Agent resides, from which to collect data from and select the *IBM Tivoli Monitoring* item from the list of available data sources (Figure 8-3 on page 215).

On the *Tivoli Enterprise Monitoring Server* tab, specify the monitoring server that you want to use to capture resource monitoring data (Figure 8-18):

- ▶ Type the IP address or the fully qualified host name of the monitoring server in the Host field on the Tivoli Enterprise Monitoring Server tab. This is different from the Host field at the top of the Create and manage configurations wizard, which indicates the monitoring agent.
- ▶ Type the user name and password for the monitoring server in Authentication.
- ▶ Change the Connection information if needed. Typically, your Tivoli system administrator will specify this information.

- ▶ Select *Save Password* to save your password locally. If you do not save your password, you might be prompted for it (depending on the host system configuration) when editing the configured location or when running test schedules that use the location.

After you have specified the monitoring server, you can select resources to capture. If the host is not managed by the monitoring server, you will see an error message.

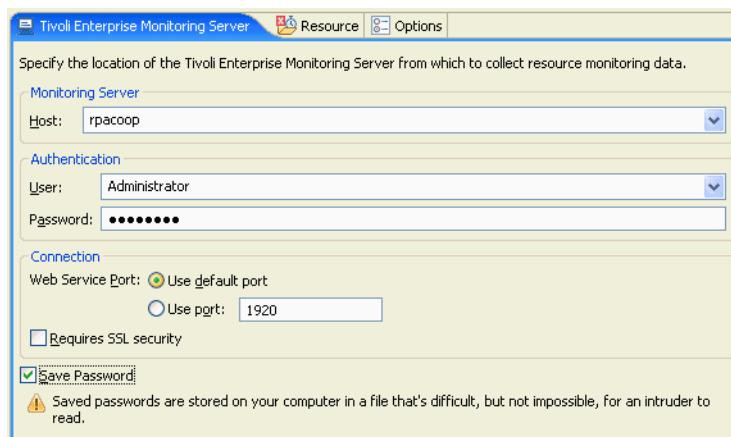


Figure 8-18 IBM Tivoli Monitoring Tivoli Enterprise Monitoring Server tab

On the *Resource* page, select the type of data that you want to capture (Figure 8-19). The tree view shows the host and all of its available IBM Tivoli Monitoring agents, and their respective counter groups and counters.

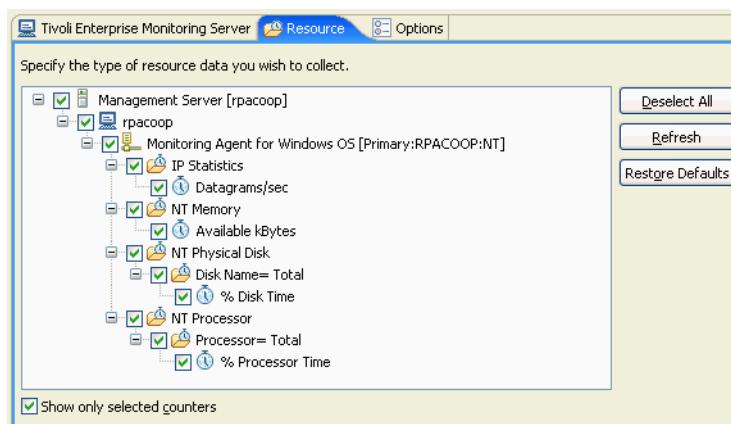


Figure 8-19 IBM Tivoli Monitoring Resource tab

Clearing *Show only selected counters* allows you to see all available counters (Figure 8-20). Be selective; monitoring all possible resource data requires substantial amounts of memory. Hover over a counter with your mouse to see details about what that counter measures.

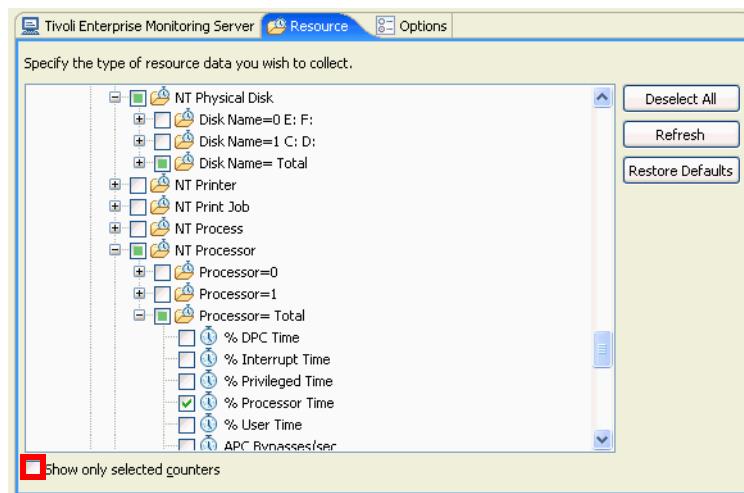


Figure 8-20 IBM Tivoli Monitoring Resource tab showing all available counters

On the *Options* tab, the time interval properties can be configured (Figure 8-21):

- ▶ Type the Polling Interval in seconds, for collecting resource data. For example, if you accept the default of 5 seconds, counter information will be collected at 5-second intervals from the specified host during the schedule run.
- ▶ Type the Timeout Interval in seconds. If the resource monitoring host does not respond within this amount of time during a schedule run, an error is logged.

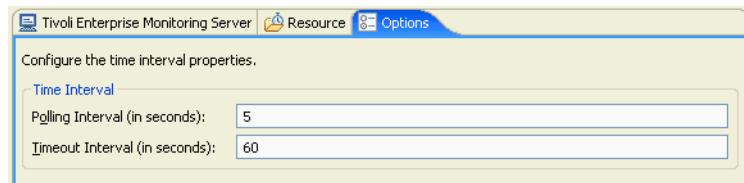


Figure 8-21 IBM Tivoli Monitoring Options tab

When the configuration is complete and saved, the resource monitoring location will be added to the performance schedule (Figure 8-22).

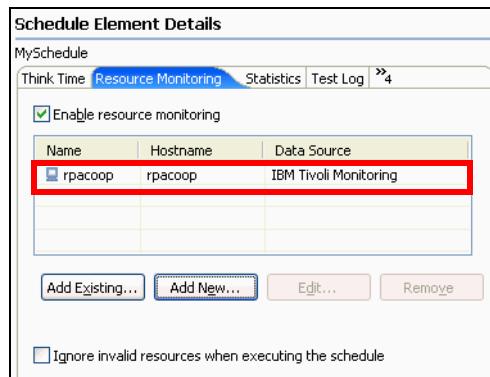


Figure 8-22 IBM Tivoli Monitoring configured for resource monitoring

8.5.5 Profiling for real-time observation

Performance Tester has the ability to collect from ITM monitoring agents, in real-time, without having to configure a resource location through a performance schedule. This capability is valuable for large enterprise environments where different machines are used for performance testing and system monitoring. In addition, even though this resource data is collected outside of a performance schedule, you can overlay these counters using the Performance Test Runs view of an existing Performance Tester Report.

To initiate real-time data collection from an ITM monitoring agent, select *Run* → *Profile* or use the toolbar button to open the launch configuration dialog (Figure 8-23). This profile action will ask you to switch into the Profile and Logging perspective of the Performance Tester product. This perspective enables profiling tools and has a different layout of the views on the workbench.



Figure 8-23 Profile launch configuration action

Resource monitoring configuration using the performance schedule is similar to configuring real-time monitoring (Figure 8-24). One difference is that you can specify where the collected data should be stored using the Destination tab. Clicking *Profile* after the tab pages have been completed will initiate data collection.

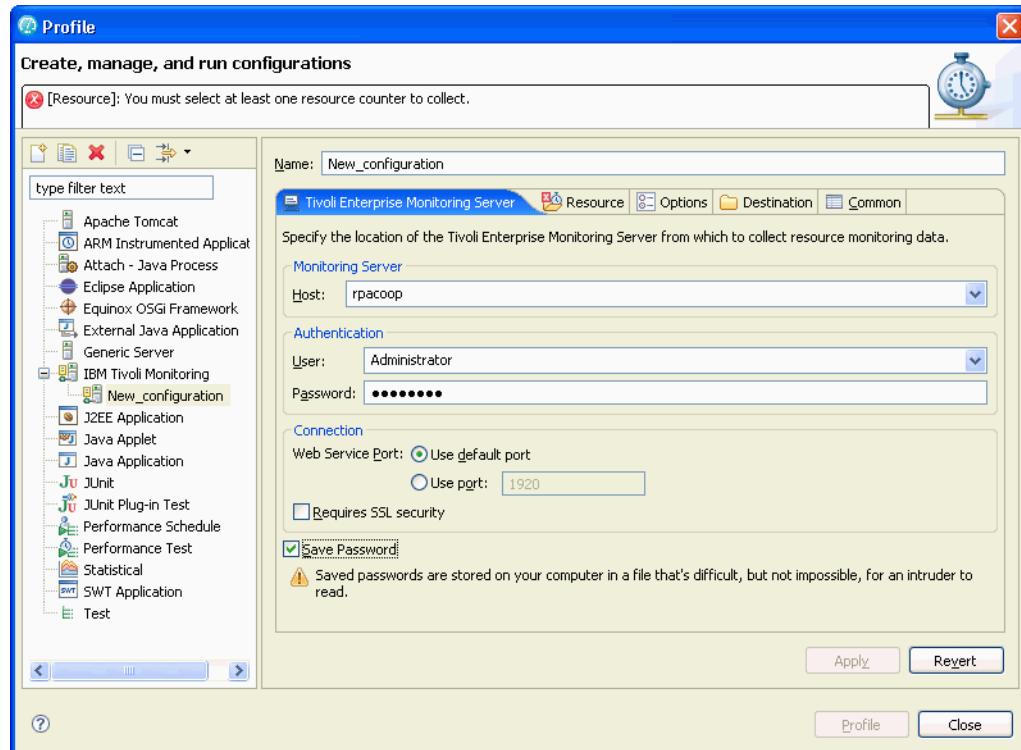


Figure 8-24 Create and run a real-time IBM Tivoli Monitoring configuration

As data is collected from the selected ITM Monitoring Agents, it is plotted on a graph display (Figure 8-25). As each new data point is observed and received by the client, the point will be plotted. The graph display is available by right-clicking on the agent in the Profiling Monitor view. Figure 8-25 illustrates the trend between the network traffic, disk activity, and processor activity.

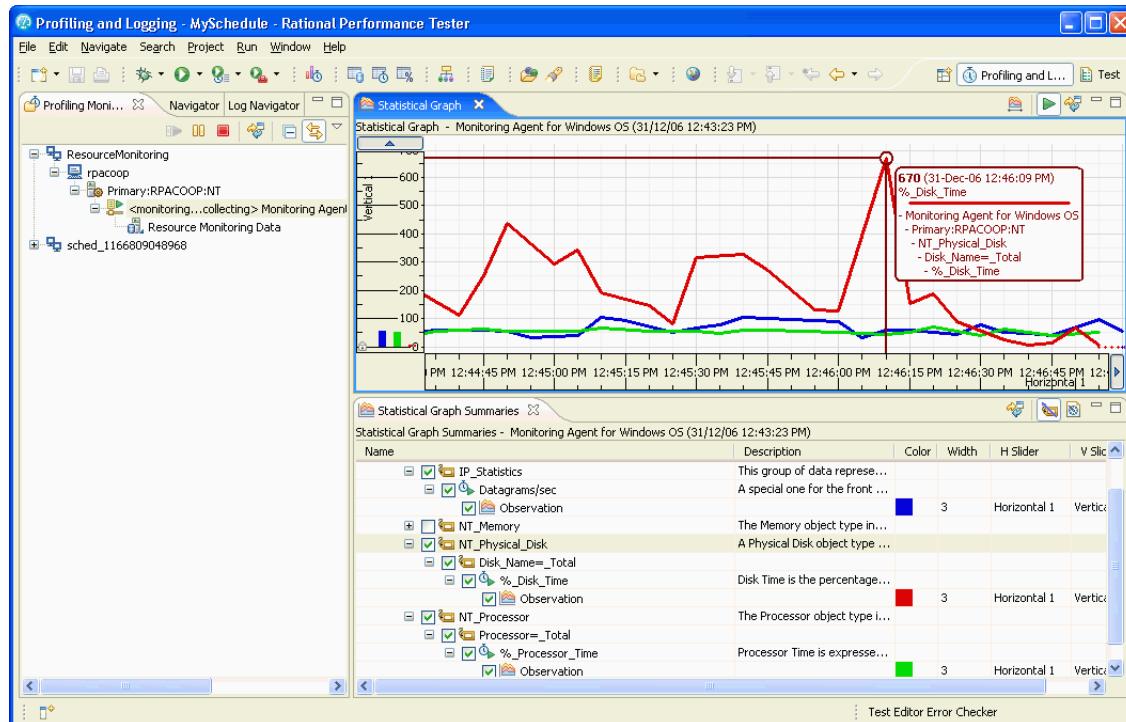


Figure 8-25 Analyze statistical resource information collected in real-time using IBM Tivoli Monitoring

8.5.6 Importing data from IBM Tivoli Monitoring

IBM Tivoli Monitoring is a managed solution that store all observed monitoring data in a data warehouse. This architecture allows you to query ITM for historical data from a particular agent, if needed.

In the workbench client, you can import the data using two methods:

1. An import wizard that is accessible by selecting *File → Import* (Figure 8-26).
2. An import wizard that is accessible by right-clicking on a performance report in the Performance Test Runs view (Figure 8-27) or on the resources report, which is the last tab in the view.

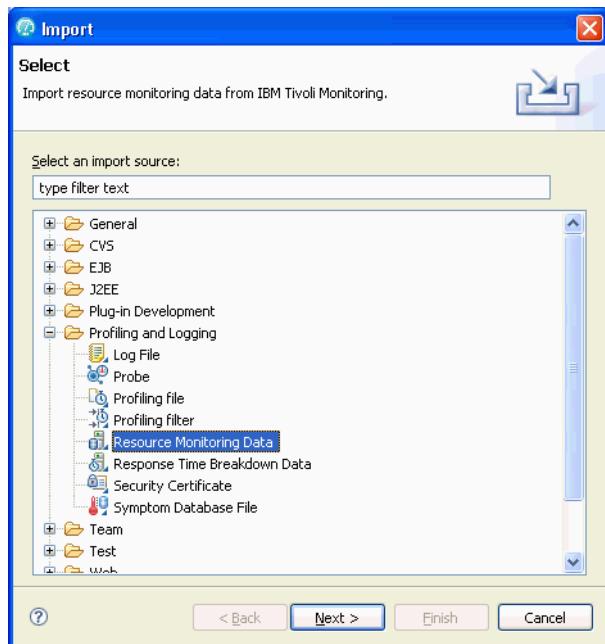


Figure 8-26 Performance Tester import wizard

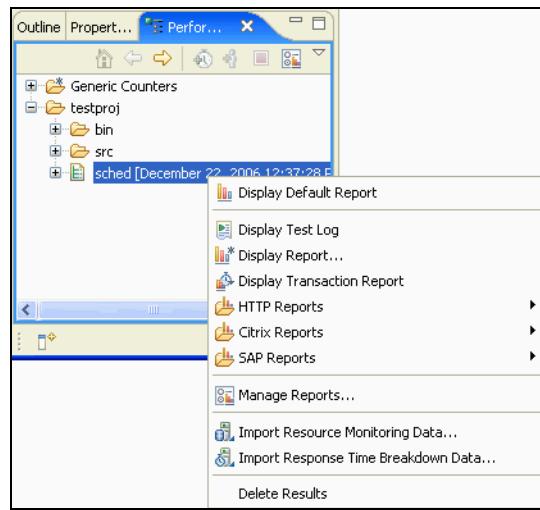


Figure 8-27 Performance result context menu

When the wizard is loaded, you must specify a resource location configured using the ITM data source (Figure 8-28).

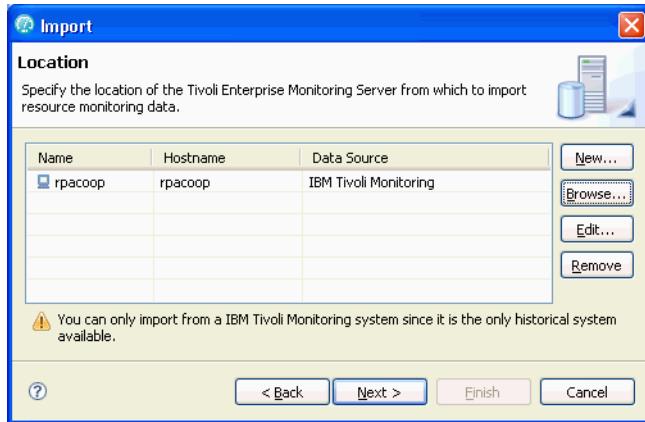


Figure 8-28 Resource Monitoring import wizard

From the three data collectors offered in Performance Tester, only ITM is capable of collecting historical data. If you try to add a resource location configured for any other data source, the collection for these locations will be ignored during import and the user will be alerted (Figure 8-29).

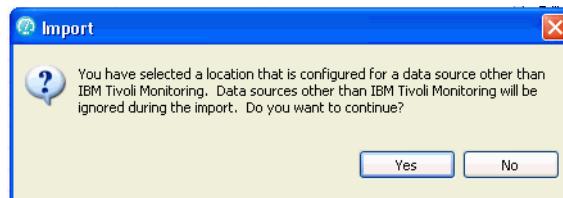


Figure 8-29 Warning indicating that only historical data sources can be used for importing data

The next stage of the import wizard is to specify the time period from which to import data (Figure 8-30). This can be specified as either the start and end times or as the amount of time you specify, counted backward from the time that you click *Finish*. You might need to adjust the time interval to compensate for clock discrepancies between the workbench and the monitoring server.

Note: When specifying the time in a specific number of units, note that, for consistency, *month* is defined as 30 days and *year* is defined as 365 days. *Days* refers to 24-hour periods, not calendar days. The units of time selected are subtracted from the point in time when you click *Finish* to import the data to give the start time. For example, if you select 2 months, the time period will be the 60 days (24-hour time periods) immediately prior to the point in time that you click *Finish*.

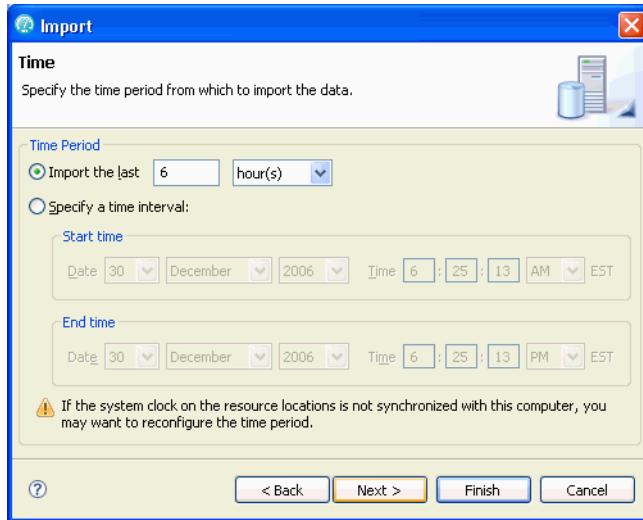


Figure 8-30 Resource Monitoring import constraints

To import data from ITM, the ITM monitoring agent itself must be configured for historical data collection. If the agent is not properly configured, an error message will be displayed to the user when attempting the import (Figure 8-31).

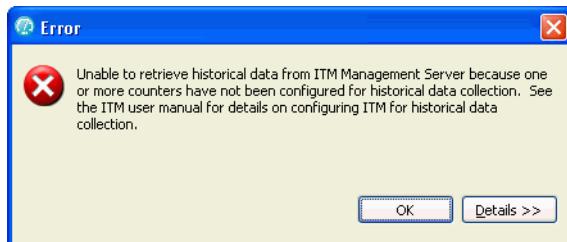


Figure 8-31 Error that ITM Monitoring Agent is not configured for historical data collection

After the import process has completed, the data is displayed in one of two ways, depending on how the import wizard was invoked:

1. If the user invoked the wizard from the *File → Import* menu, this action indicates that there is no association to a performance result and the data is displayed using the default statistical view in the Profile and Logging perspective (Figure 8-32).
2. If the user invoked the wizard by right-clicking on a performance result, this action indicates the imported data is associated to the selected performance result. The data can be viewed using the Resource tab in the default report when the performance result is opened.

Alternatively, if you manually switch to the Profile and Logging perspective, you can view and analyze the data using the default statistical view in the Profile and Logging perspective.

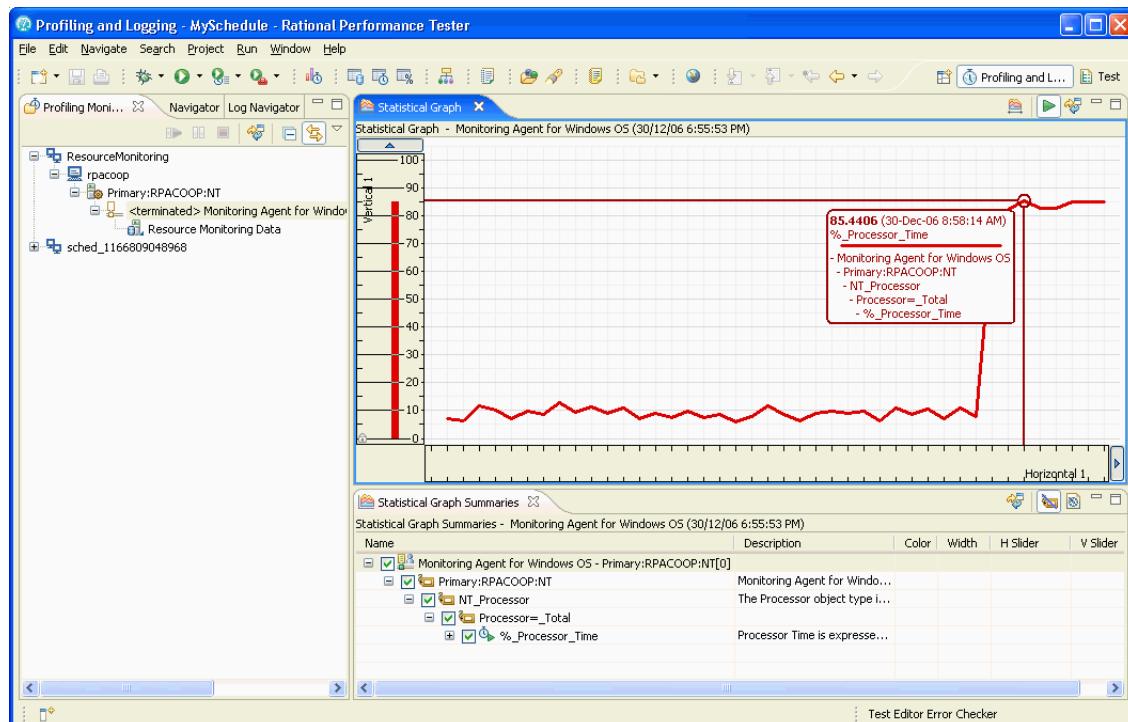


Figure 8-32 Display and analyze statistical resource information imported from IBM Tivoli Monitoring

8.6 Customizing reports and overlaying counters

Performance Tester's report infrastructure allows for display of rich interactive graphs and tables. To aid in problem determination, each graph in the report infrastructure has been enhanced to allow resource data to be placed over the already rendered graphs. This action allows one to correlate two separate data sets, such as page response time and % Processor Time. Correlation of resource data with response time data help an analyst to determine if there is an issue with the system, capacity of the environment, or the application itself.

There are two ways to overlay counters on a report:

1. Add performance counters to an existing graph.
2. Drag-and-drop counters onto an existing report.

Add performance counters to an existing graph

Right-click on an existing graph and select *Add/Remove Performance Counters* → *Resource Monitoring Counter* (Figure 8-33) to display the Resource Counters configuration dialog. This menu item is available from any graph in the Performance Report view.

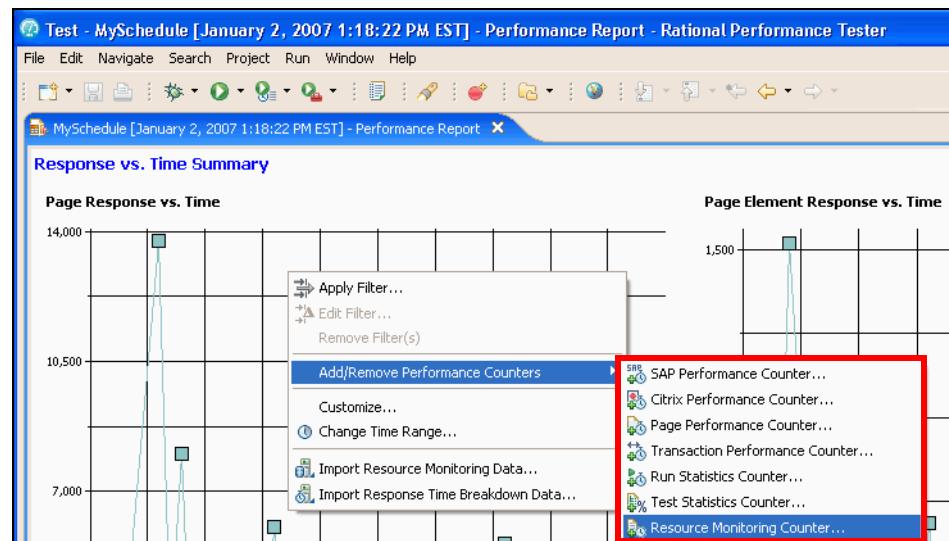


Figure 8-33 Overlay resource monitoring counter on an existing report

The Resource Counters configuration dialog (Figure 8-34) allows you to select specific counters and adjust the scaling factor for each counter. Adjusting the scaling factor for a counter is important when trying to analyze correlation between counters that have different data types. For example, processor time is a percentage quantity where as memory is measured in bytes.

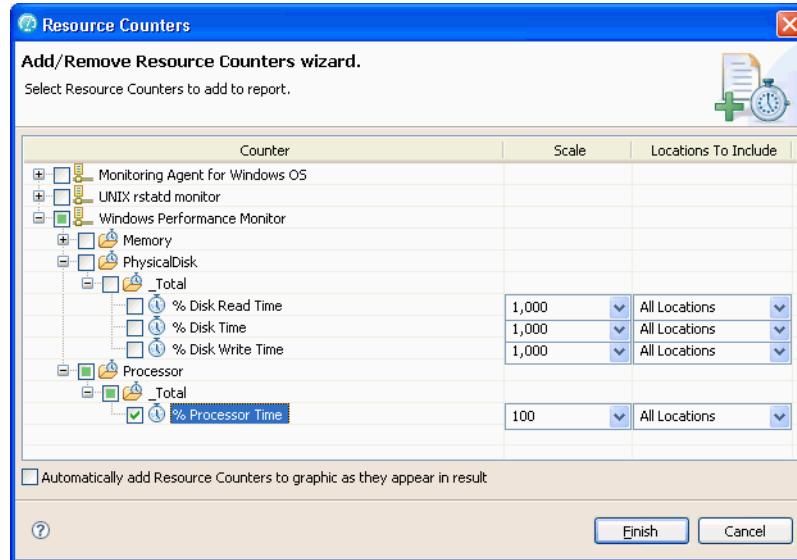


Figure 8-34 Resource Counters configuration dialog

When the counters are selected and the configuration dialog is complete, the counters are drawn on the selected graph (Figure 8-35). Hovering over any data point using the mouse will provide a tool-tip containing information about the data point, such as the value of the observation.

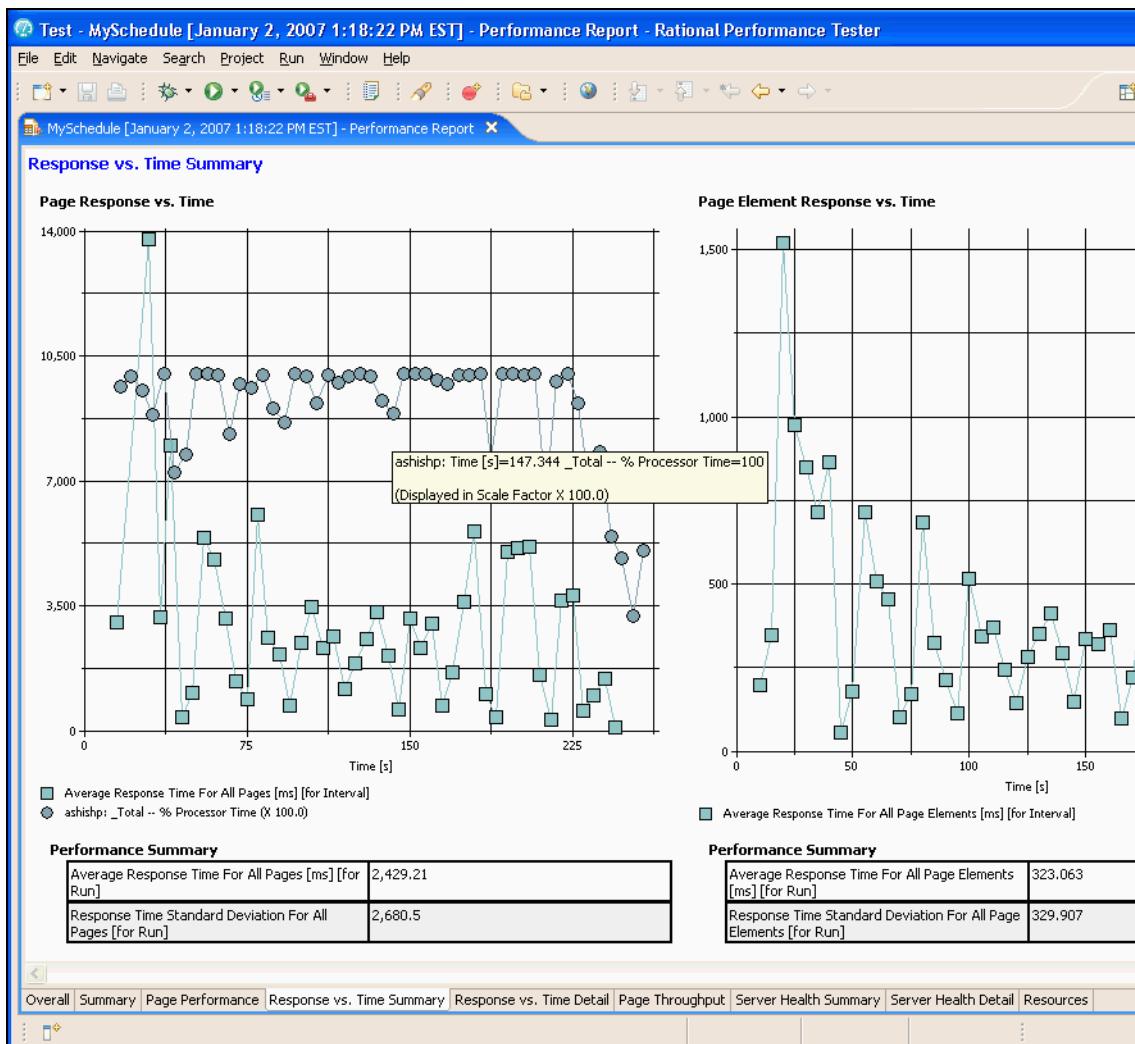


Figure 8-35 Response versus Time summary correlated with the % Processor Time resource counter

Drag-and-drop counters onto an existing report

Drag-and-drop counters from the Performance Test Run view onto an existing report (Figure 8-36). Expanding the tree control in this view allows the user to select specific counters. The tree is organized by the host name, under which the data collection agents are listed, such as UNIX rstatd monitor.

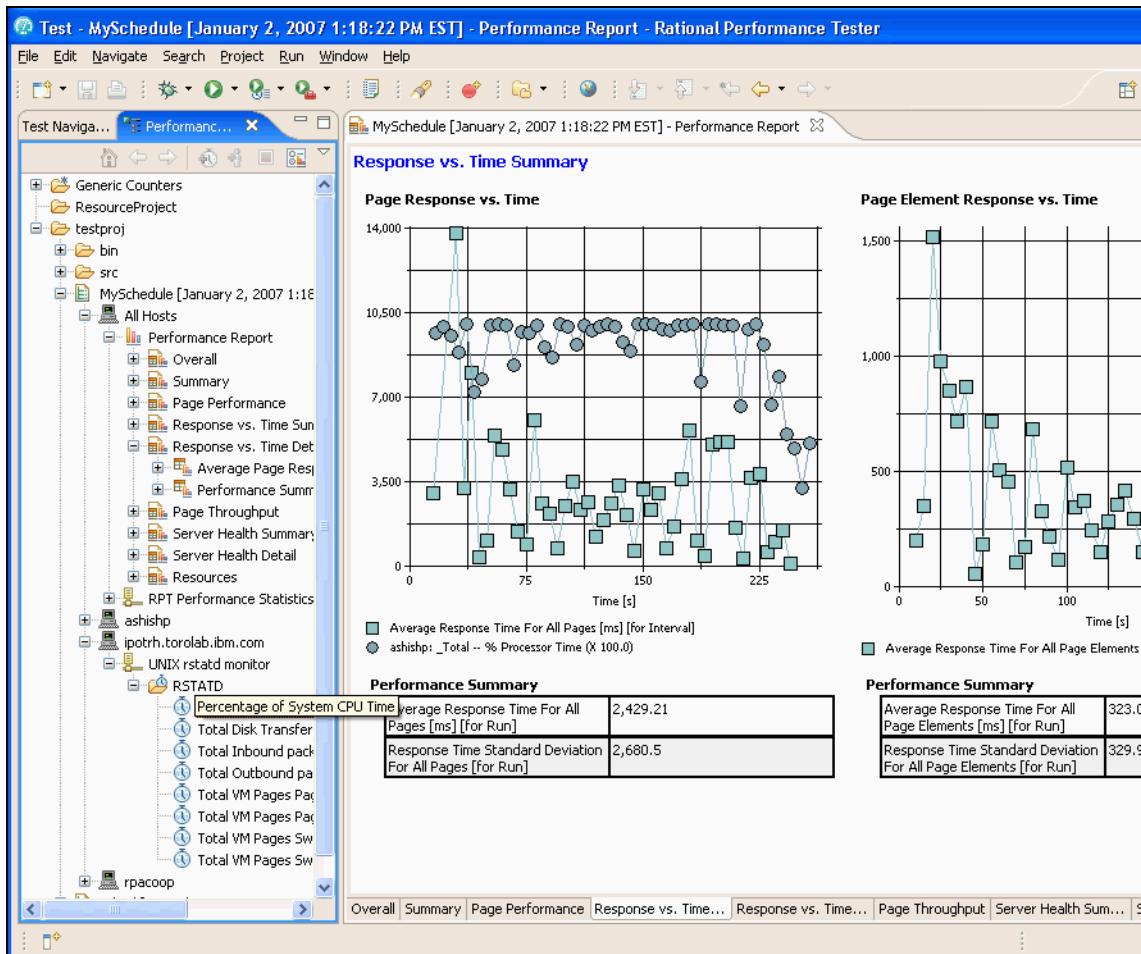


Figure 8-36 Drag-and-drop a resource counter from the Performance Test Run view

A suggested report to modify with user-defined counters is the Resources report, the last tab in the Performance Report view (Figure 8-37). This report spans the entire view allowing more working area than other reports in the view.

The layout of the graph can also be customized using the *Customize* menu item when you right-click on the graph. This action allows you to set the X and Y axis limits, the legend, and change the time range.

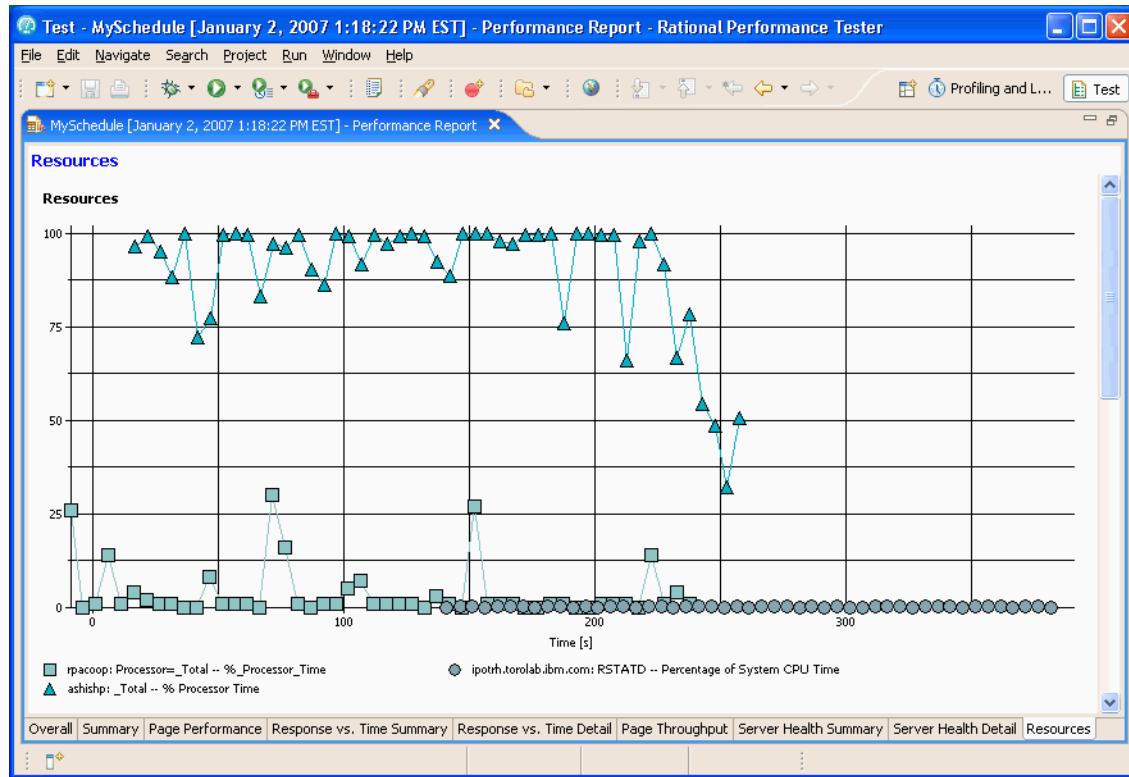


Figure 8-37 Resources report in a performance report

8.7 More information

For more information, consult these resources:

- ▶ Microsoft Developer Network (MSDN®), Performance Monitoring:
<http://msdn2.microsoft.com/en-gb/library/aa830465.aspx>
- ▶ Eclipse Test & Performance Tools Platform documentation:
<http://www.eclipse.org/tptp>
- ▶ *IBM Tivoli Monitoring v6.1.0: Installation and Setup Guide* (First Edition, November 2005), GC32-9407
<http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/topic/com.ibm.help.ic.doc/welcome.htm>
- ▶ Administering OMEGAMON Products: CandleNet Portal, V195 July 2004
Candle® Corporation
- ▶ RSTAT RFC 1831,1832,1833 (these are the RFCs consulted when implementing the RSTAT monitoring component):
<http://rfc.net/rfc1831.html>
<http://rfc.net/rfc1832.html>
<http://rfc.net/rfc1833.html>
<http://docs.hp.com/en/B2355-60103/rstat.3N.html>



Application monitoring

This chapter explains why application monitoring is important in performance testing and shows how to enable this monitoring using Performance Tester. We describe the technology that enables application monitoring, how to configure your environment, and how to leverage the numerous visual interfaces when analyzing your application.

We cover the following topics:

- ▶ Overview of application monitoring
- ▶ End-to-end distributed business transactions
- ▶ Application response measurement
- ▶ Configuring your environment
- ▶ Enabling real-time application monitoring
- ▶ Collecting application monitoring data from IBM Tivoli family of products

9.1 Overview of application monitoring

Applications executing in a production environment need to be managed to ensure that service level agreements (SLAs) are met. Application management is one of the fastest growing areas, representing 24% of the infrastructure performance management market. Over the last decade, Java2 Enterprise Edition (J2EE) platform for building applications has grown to become a very large slice of the enterprise. Today, there are J2EE-based application and services that help to integrate internal and external components of a business' Information Technology (IT) infrastructure. In addition, these components exist because of various business processes that depend on these applications and services fueling the process. Therefore, ensuring availability, capacity, and performance of an application is crucial to a business' operation.

Monitoring transactions end-to-end is crucial for problem determination and analysis in enterprise server environments, as these environments are becoming more and more complex each day. Being able to understand how a particular transaction is behaving in a distributed environment during load testing or functional testing enables application developers to more easily fix the problem. In addition, integration with the IBM Tivoli Composite Application Manager (ITCAM) family of products gives customers the ability to diagnose problems that have already occurred in production, giving application developers access to production data in the development environment in a format that is understood by developers.

9.1.1 End-to-end distributed business transactions

Monitoring applications is not just as simple as profiling them. In today's complex world, a convergence is present to coupling various applications and services into bundles of composite applications that achieve a business objective or process. Composite applications involve numerous application platforms and hardware resources, and thus, creating a distributed environment (Figure 9-1).

Application monitoring across these growing complex distributed environments requires tracing of transactions through the entire environment. Tracing a transaction from the originator, into the various layers of the application and software platforms involved, across physical systems, and between various software products provides enough information to decompose it. Decomposition of a transaction is important in determining where a bottleneck or performance degradation is occurring in the environment—in turn, providing a means for problem determination in a distributed environment.

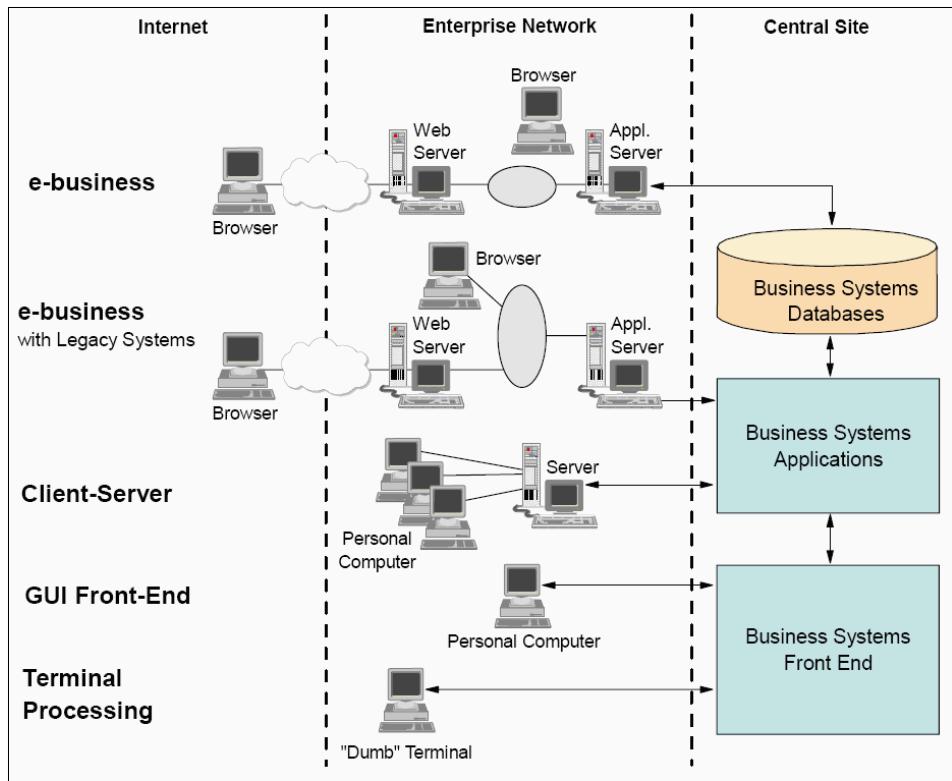


Figure 9-1 Distributed environment complexity

An end-to-end transaction is composed of a large number of individual, interrelated transactions (Figure 9-2). A single transaction can be responsible for completing a business process or broker information between disparate systems across applications or services. If any of these transactions has poor performance or is unavailable at any given time, this affects the overall user experience and business objective.

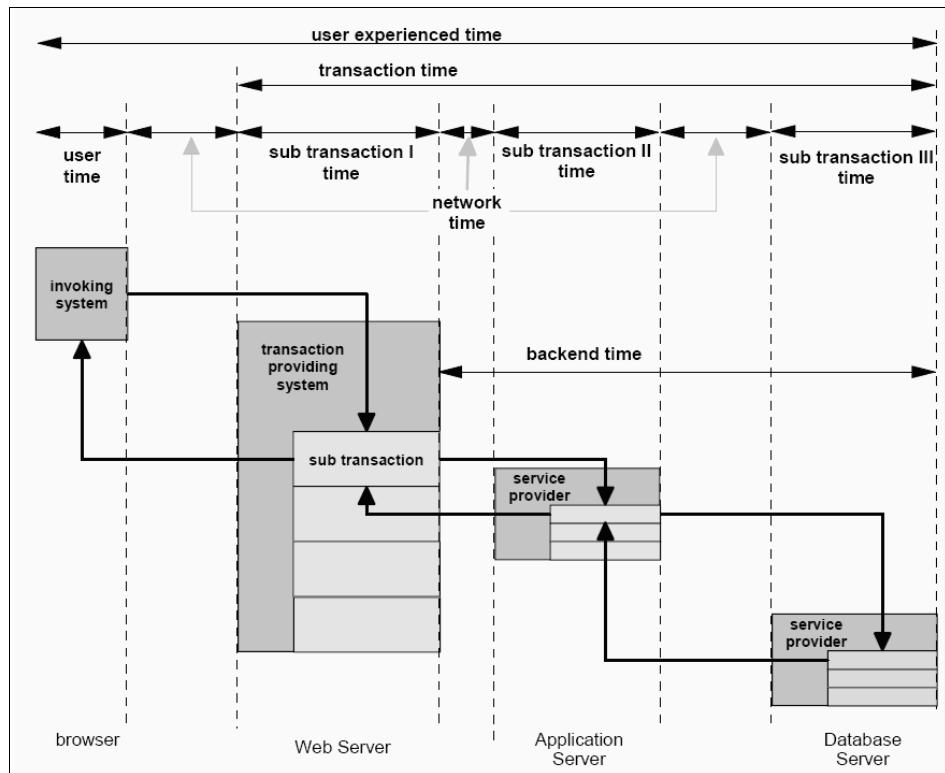


Figure 9-2 Transaction breakdown

Measuring the response time for a transaction from end-to-end is known as the *response time breakdown* (RTB) feature in Performance Tester. Response time breakdown allows customers to monitor their J2EE applications in real-time as they are executed in a distributed application server environment.

9.1.2 Application response measurement

The application response measurement (ARM) standard is developed by The Open Group, which is a vendor- and technology-neutral consortium that strives to enable access to integrated information within and between enterprises based on open standards and global interoperability. The ARM standard describes a common method for integrating enterprise applications as manageable entities. The ARM standard allows users to extend their enterprise management tools directly to applications, creating a comprehensive end-to-end management capability that includes measuring application availability, application performance, application usage, and end-to-end transaction response time.

Applications make calls to application programming interfaces (APIs) defined by The Open Group when transactions begin and end, allowing these transactions to be measured and monitored. This information can be used to support service level agreements and to analyze response time across distributed systems.

ARM instrumentation has grown in popularity since it was first introduced in 1996 and is now built into software from leading vendors such as IBM, Hewlett-Packard, SAS, and Siebel. Software that is not already ARM-enabled can be made to have calls to the ARM API embedded directly in the source code, or calls can be inserted into the machine code at run-time.

Transaction correlation

To correlate transactions that occur on different processes and machines in a distributed environment, the ARM standard calls for the use of an ARM correlator. A correlator is generated for every root (or edge) transaction and each of its child transactions. A business transaction is determined by building a tree using these correlators. This process allows the ARM implementation to trace the path of a distributed transaction through the infrastructure.

In distributed J2EE environments, one method to transport the ARM correlator across different application servers is to embed it a part of a Common Object Request Broker Architecture (CORBA) Portable Interceptor when a distributed call is executed. Portable Interceptors are objects that an Object Request Broker (ORB) invokes in the path of an operation invocation to monitor or modify the behavior of the invocation transparently. For our purposes, a Portable Request Interceptor is created and used to marshal and de-marshal the correlator during distributed transactions. The user is required to install the request interceptor in their application server.

ARM implementation

Performance Tester leverages an existing ARM implementation present in the IBM Tivoli Composite Application Manager (ITCAM) for Response Time Tracking product, which was formerly known as IBM Tivoli Monitoring for Transaction Performance (TFTP).

The Tivoli ARM engine is a multi-threaded application implemented as the tapmagent (tapmagent.exe on Windows based platforms). The ARM engine exchanges data though an IPC channel, using the libarm library (libarm32.dll on Windows based platforms), with ARM instrumented applications. The data collected is then aggregated in order to generate useful information, correlated with other transactions, and thresholds are measured based upon user requirements. This information is then rolled up to the management server and placed into the database for reporting purposes.

The Tivoli transaction monitoring infrastructure allows notifying the Tivoli ARM engine as to which transactions need to be measured. This behavior allows for monitoring a specific type of transaction, such as transactions that invoke a Servlet; or monitoring a single transaction end-to-end. In addition, transactions can be monitored when a specific user-defined threshold has been violated.

Correlation using the Tivoli ARM engine is possible using four global unique identifiers (GUID):

- ▶ Origin host UUID—Indicates the ID of the physical host where the transaction originated
- ▶ Root transID—Indicates the ID where of the transaction originated
- ▶ Parent transID—Indicates the ID of a transaction that has sub-transactions
- ▶ Transaction classID—Indicates the ID of any transaction

For example, a single transaction can be composed of several sub-transactions that are related using these GUIDs (Figure 9-3). This illustration shows how each sub-transaction is related to its parent, all the way up to the root of the entire transaction. Typically, the root transaction in the Internet environment is considered to be the URL itself.

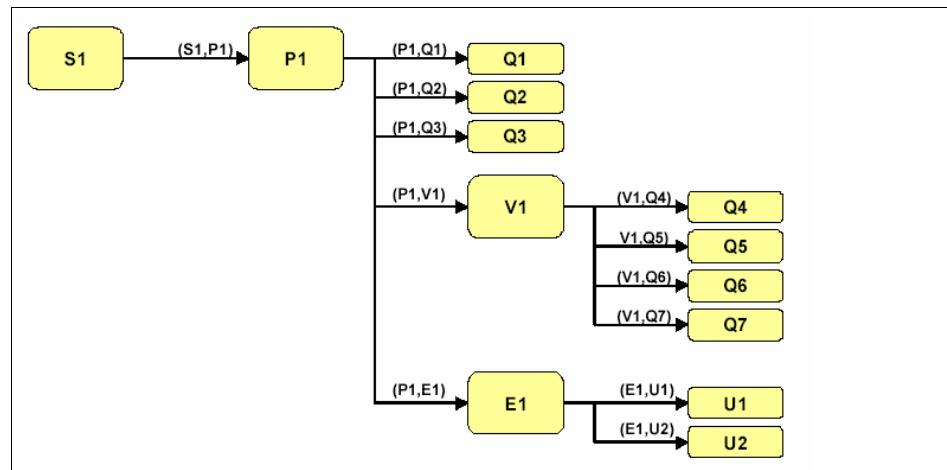


Figure 9-3 Correlating ARM transactions with their sub-transactions

When all the transactions are observed at the Tivoli ARM engine, then they are sent to the Management Server, where they are correlated and persisted.

The Tivoli ARM engine has a plug-in system by which a third-party applications can register their application with the engine through a well-known Java interface. When registered, the application will receive notifications of ARM events and, if desired, the entire ARM objects as well. This interface is known as the ARM plug-in interface. This interface enables applications to process ARM events in real-time, rather than waiting for the events to be available at the management server.

Furthermore, the interface also allows applications to append data to the ARM correlator that is sent from one machine to another. Although this can be dangerous, the total number of bytes should never exceed the size of the correlator according to the ARM specification, or there might be adverse effects in the environment. The Tivoli ARM engine does not manage any data that is appended to the ARM correlator, and therefore, it is up to the third-party application developer to ensure stability of any modification to the original ARM correlator generated by the engine.

9.1.3 Instrumentation

To monitor an application using ARM, a software developer must insert snippets of application code into it, also known as a *probe*. These probes make calls to the Open Group ARM standard API, and when the application (with the snippets) is executed the ARM implementation will process these calls, which is also known as a transaction. The act of inserting the probe into the application intended to be monitored is known as *instrumentation*, because it behaves as the response time *measurement device* when the application is executing.

Often, having a software developer manually modify application code to insert the probe can be cumbersome and might increase the overhead in maintaining the application code itself. Various technologies have been developed in industry to help automate this process. One mechanism that is widely used is known as *byte-code instrumentation*. This mechanism involves a process by which the application source code is compiled into sets of byte-codes (which are intended to be executed by virtual machine) and byte-codes representing the probes are inserted before and after specific locations in the set of byte-codes representing the original application.

J2EE monitoring component

Performance Tester leverages the Tivoli J2EE Monitoring Component to support byte-code instrumentation for IBM WebSphere Application Server and BEA WebLogic Application Server. The Tivoli J2EE monitoring is provided by Just In Time Instrumentation (JITI).

JITI allows managing of J2EE applications that do not provide system management instrumentation by injecting probes at class-load time, that is, no application source code is required or modified in order to perform monitoring. Additionally, the probes can easily be turned on and off as required. This is an important difference, which means that the additional transaction decomposition can be turned on only when required. It is important that this capability is available as though this component has low overheads (all performance monitoring has some overhead; the more monitoring you do, the greater the overhead). The fact that J2EE monitoring can be easily enabled and disabled from the user is a powerful feature.

With the release of JDK™ 1.2, Sun™ Microsystems included a profiling mechanism within the JVM™. This mechanism provided an API that could be used to build profilers called JVMPPI, or Java Virtual Machine Profiling Interface. The JVMPPI is a bidirectional interface between a Java virtual machine and an in-process profiler agent. JITI uses the JVMPPI and works with un-instrumented applications. The JVM can notify the profiler agent of various events, corresponding to, for example, heap allocation, thread start, and so on. Or the profiler agent can issue controls and requests for more information through the JVMPPI, for example, the profiler agent can turn on/off a specific event notification, based on the needs of the profiler front end.

JITI starts when the application classes are loaded by the JVM (for example, the IBM WebSphere Application Server). The injector alters the Java methods and constructors specified in the registry by injecting special byte-codes in the in-memory application class files (Figure 9-4). These byte-codes include invocations to hook methods that contain the logic to manage the execution of the probes. When a hook is executed, it gets the list of probes currently enabled for its location from the registry and executes them. JITI probes make ARM calls and generates correlators in order to allow sub-transactions to be correlated with their parent transactions.

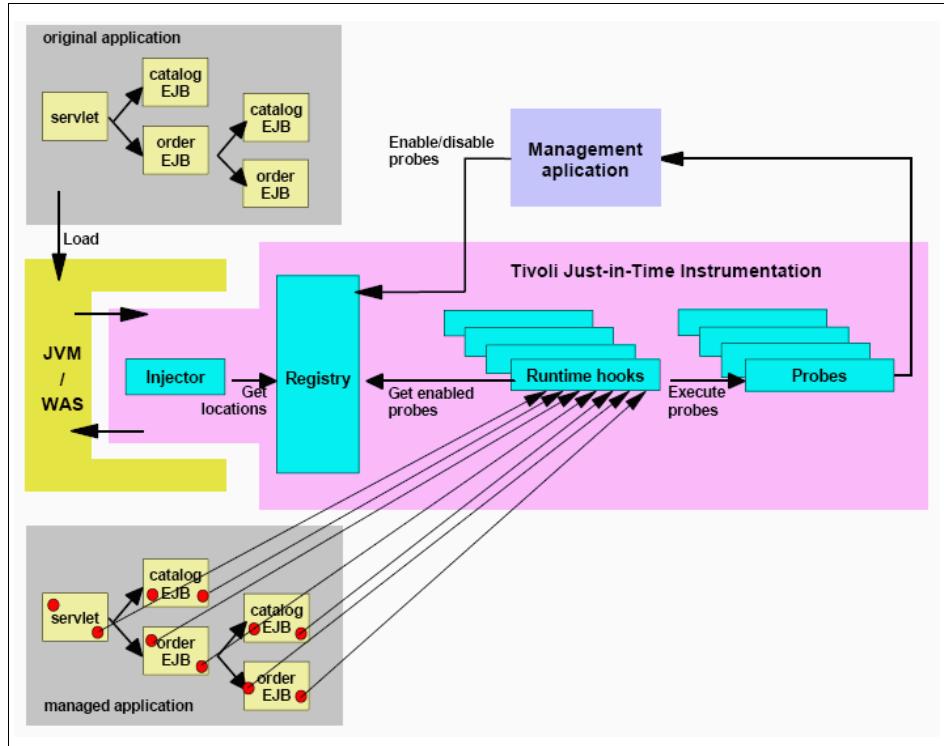


Figure 9-4 Tivoli Just-In-Time Instrumentation overview

The J2EE monitoring component supports instrumenting the following J2EE aspects:

- ▶ Servlets
- ▶ Entity beans
- ▶ Session beans
- ▶ JMS
- ▶ JDBC™
- ▶ RMI-IIOP
- ▶ J2C
- ▶ Web services

9.2 Configuring your environment

In this section we discuss how a J2EE environment is configured for application monitoring using the Tivoli J2EE Monitoring Component. This component supports IBM WebSphere Application Server and BEA WebLogic Application Server (WLS). It is important to understand and be aware of the modifications required to the application server environment. Also, Performance Tester includes a helpful tool called the Application Server Instrumenter (ASI) that will help automate instrumentation.

9.2.1 IBM WebSphere Application Server

For JITI to start when the application classes are loaded by the Application Server JVM, modifications to the boot class path of the server instance are required. The boot class path allows a series of Java archives (JARs) to be loaded into the JVM runtime during execution immediately as the JVM is being loaded. In the case of WebSphere Application Server, there are three files that are modified:

```
server.xml  
variables.xml  
pmirm.xml
```

The `server.xml` and `variables.xml` files are located at:

- ▶ WebSphere Application Server v5.x:
`{WAS_HOME}/config/cells/{cell_name}/nodes/{node_name}/servers/{server_name}`
- ▶ WebSphere Application Server v6.x
`{WAS_PROFILE_HOME}/config/cells/{cell_name}/nodes/{node_name}/servers/{server_name}`

Whereas, `pmirm.xml` is located at:

- ▶ WebSphere Application Server v5.x
`{WAS_HOME}/config/cells/{cell_name}`
- ▶ WebSphere Application Server v6.x
`{WAS_PROFILE_HOME}/config/cells/{cell_name}`

Note: Notice that the only difference between version 5 and 6 in these paths are the starting anchor directories. In version 6, the server product introduced a notion of profiles that can be placed in locations or partitions other than where the product itself is installed. The location where the profile is stored is known as the `WAS_PROFILE_HOME` directory, whereas, `WAS_HOME` is where the product is installed.

variables.xml

The variables.xml file consists of file system path information. This path information is used to define several variables that are used in the server.xml file. The variables.xml file might look like the following XML excerpt:

```
<variables:VariableMap xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:variables="http://www.ibm.com/websphere/appserver/schemas/5.0/
        variables.xmi" xmi:id="VariableMap_1">
    <entries xmi:id="VariableSubstitutionEntry_1166740318828"
        symbolicName="SERVER_LOG_ROOT" value="${LOG_ROOT}/server1"
        description="The log root directory for server server1."/>
    <entries xmi:id="VariableSubstitutionEntry_1166740318891"
        symbolicName="WAS_SERVER_NAME" value="server1"
        description="Name of the application server."/>
    <entries xmi:id="VariableSubstitutionEntry_1166804719953"
        symbolicName="MA_LOG_QUALDIR" value="J2EE/server1_100"
        description="Base directory for logging for the TMTP Management
            Agent"/>
    <entries xmi:id="VariableSubstitutionEntry_1166804720094"
        symbolicName="MA" value="C:/PROGRA~1/IBM/SDP70/DCI/rpa_prod/TIVOLI~1"
        description="Base installation directory of the TMTP Management
            Agent"/>
    <entries xmi:id="VariableSubstitutionEntry_1166804720500"
        symbolicName="MA_LIB"
        value="C:/PROGRA~1/IBM/SDP70/DCI/rpa_prod/TIVOLI~1/lib"
        description="Management Agent library directory"/>
    <entries xmi:id="VariableSubstitutionEntry_1166804720672"
        symbolicName="MA_INSTRUMENT"
        value="C:/PROGRA~1/IBM/SDP70/DCI/rpa_prod/TIVOLI~1/app/instrument/61"
        description="Base installation directory of the TMTP J2EE
            Instrumentation application."/>
    <entries xmi:id="VariableSubstitutionEntry_1166804720828"
        symbolicName="MA_INSTRUMENT_LIB"
        value="C:/PROGRA~1/IBM/SDP70/DCI/rpa_prod/TIVOLI~1/app/instrument/
            61/lib"
        description="TMTP J2EE Instrumentation application library
            directory."/>
    <entries xmi:id="VariableSubstitutionEntry_1166804721031"
        symbolicName="MA_INSTRUMENT_APPSERVER_CONFIG"
        value="C:/PROGRA~1/IBM/SDP70/DCI/rpa_prod/TIVOLI~1/app/instrument
            /61/appServers/server1_100/config"
        description="TMTP J2EE Instrumentation application config
            directory."/>
</variables:VariableMap>
```

server.xml

The server.xml file includes a set of Java Virtual Machine (JVM) arguments (also known as genericJVMArguments) that are loaded when WAS executes its bootstrap procedures. These arguments appear at the bottom of the XML file, in the JavaProcessDef process definitions section, and might appear similar to the following excerpt:

```
<jvmEntries xmi:id="JavaVirtualMachine_1166740318031"
    verboseModeClass="false"
    verboseModeGarbageCollection="false"
    verboseModeJNI="false"
    runHProf="false"
    hprofArguments=""
    debugMode="false"
    debugArgs="-Djava.compiler=NONE -Xdebug -Xnoagent
        -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777"
    genericJvmArguments="
        -Xbootclasspath/a:${MA_INSTRUMENT}\lib\jiti.jar;
        ${MA_INSTRUMENT}\lib\bootic.jar;${MA_INSTRUMENT}\ic\config;
        ${MA_INSTRUMENT_APPSERVER_CONFIG}
        -Dma.instrument=${MA_INSTRUMENT}
        -Dma.appserverconfig=${MA_INSTRUMENT_APPSERVER_CONFIG}
        -Dttmp.user.dir=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1
        -Dcom.ibm.tivoli.jiti.config=${MA_INSTRUMENT_APPSERVER_CONFIG}\
            config.properties
        -Dcom.ibm.tivoli.transperf.logging.qualDir=${MA_LOG_QUALDIR}
        -Dcom.ibm.tivoli.probe.directory=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod
            \TIVOLI~1\app\instrument\61\lib\probes
        -Dws.ext.dirs=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument
            \61\lib\ext -Djlog.propertyFileDir=${MA_INSTRUMENT_APPSERVER_CONFIG}
        -Xrunvirt:agent=jvmpi:C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app
            \instrument\61\appServers\server1_100\config\jiti.properties,
        agent=piAgent:server=enabled">

    <systemProperties xmi:id="Property_1166804730297"
        name="com.ibm.tivoli.jiti.injector.ProbeInjectorManagerChain.file"
        value="C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61
            \appServers\server1_100\config\injector.properties"
        description="ITCAM Primary Injector"/>

    <systemProperties xmi:id="Property_1166804733531"
        name="com.ibm.websphere.pmi.reqmetrics.PassCorrelatorToDB"
        value="false"
        description="Enables WAS to pass the ARM correlator to the database
            when 'true'."/>
</jvmEntries>
```

The server.xml file also defines a Portable Request Interceptor that is executed during Remote Method Invocation (RMI) Internet Inter-Orb Protocol (IIOP) calls, also referred to as RMI-IIOP. This interceptor is defined using a well defined interface contained in the Java SDK. The XML fragment is placed with the other interceptors in the file, and is shown here:

```
<interceptors xmi:id="Interceptor_1094820250154"
    name="com.ibm.tivoli.transperf.instr.corba.iiop.TxnPortableInterceptor"/>
```

pmirm.xml

The pmirm.xml file makes changes to the WAS PMI Request Metrics configuration by enabling the metrics.

9.2.2 BEA WebLogic Application Server (WLS)

Similar modifications are required to WLS as made in WebSphere Application Serve. In the case of WLS, there are two files that are modified:

```
start<server_name>Server (for example: startMedRecServer)
commEnv
```

start<server_name>Server

The start<server_name>Server script file starts the WebLogic server. The script is modified to include a set of Java Virtual Machine (JVM) arguments. These arguments might appear similar to the following excerpt:

```
@rem Begin TMTP AppID MedRecServer_100

set PATH=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61
    \lib\windows;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument
    \61\lib\windows\sjiti;%PATH%

set MA_INSTRUMENT=
    C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61
set JITI_OPTIONS=
    -Xbootclasspath/a:%MA_INSTRUMENT%\lib\jiti.jar;
    %MA_INSTRUMENT%\lib\bootic.jar;%MA_INSTRUMENT%\ic\config;
    C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\INSTRU~1\61\APPSE~1
    \MEDREC~1\config
    -Xrunjvmp: C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\INSTRU~1
        \61\APPSE~1\MEDREC~1\config\jiti.properties
    -Dtmtcp.user.dir=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1
    -Dcom.ibm.tivoli.jiti.probe.directory=C:\PROGRA~1\IBM\SDP70\DCI
        \rpa_prod\TIVOLI~1\app\INSTRU~1\61\lib\probes
    -Dma.instrument=%MA_INSTRUMENT%
    -Dma.appserverconfig=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app
        \INSTRU~1\61\APPSE~1\MEDREC~1\config
```

```

-Dcom.ibm.tivoli.jiti.config=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1
    \app\INSTRU~1\61\APPSER~1\MEDREC~1\config\config.properties
-Dcom.ibm.tivoli.transperf.logging.qualDir=J2EE\MedRecServer_100
-Dweblogic.TracingEnabled=true
-Djlog.propertyFileDir=C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app
    \INSTRU~1\61\APPSER~1\MEDREC~1\config
-Dcom.ibm.tivoli.jiti.injector.ProbeInjectorManagerChain.file=
    C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\INSTRU~1\61\APPSER~1
    \MEDREC~1\config\injector.properties

set JAVA_OPTIONS=%JITI_OPTIONS% %JAVA_OPTIONS%

set CLASSPATH=
%CLASSPATH%;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\6
1\lib\ext\instrument.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app
\instrument\61\lib\ext\ejflt.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVO
LI~1\app\instrument\61\lib\ext\jflt.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_pr
od\TIVOLI~1\app\instrument\61\lib\ext\jffdc.jar;C:\PROGRA~1\IBM\SDP70\DC
I\rpa_prod\TIVOLI~1\app\instrument\61\lib\ext\jlog.jar;C:\PROGRA~1\IBM\SD
P70\DCI\rpa_prod\TIVOLI~1\app\instrument\61\lib\ext\copyright.jar;C:\PRO
GRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61\lib\ext\core_in
str.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61\li
b\ext\armjni.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrum
ent\61\lib\ext\epbam.jar;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app
\instrument\61\lib\ext\concurrency_util.jar

```

commEnv

In addition, the `commEnv` script file, which initializes the common environment before WebLogic starts, modifies the system PATH to include the JITI library. The modification to the script might appear similar to this:

```

@rem TMTP Begin

set PATH=
C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61\lib\window
s;C:\PROGRA~1\IBM\SDP70\DCI\rpa_prod\TIVOLI~1\app\instrument\61\lib\wind
ows\sjiti;%PATH%

@rem TMTP End

```

9.2.3 Using the Application Server Instrumenter (ASI)

The data collection infrastructure, by default, is installed on the local machine where you have installed the Rational Performance Tester V7.0 workbench. To collect data from other machines used in distributed applications, the data collection infrastructure must also be present on all of these remote hosts. You must, therefore, install the Rational Performance Tester Agent V7.0 on all remote hosts that will be used in your distributed applications.

There are two options when adding/removing instrumentation to/from an application server:

- ▶ Command line script
- ▶ Graphical user interface (GUI)

Instrumenting an application server from the command line

To use the command line to instrument an application server, follow these steps:

- ▶ Ensure that the application server to be instrumented is running.
- ▶ Open a command line and go to the rpa_prod directory under the data collection infrastructure installation directory, which is typically found in:
 - Windows: C:\Program Files\IBM\SDP70\DCI
 - Linux: /opt/IBM/SDP70/DCI
- ▶ The instrumentation utility is called instrumentServer.bat (Windows) or instrumentServer.sh (Linux). Type the command name with no arguments to see the syntax details for the command.
- ▶ Type the command name with the desired arguments to instrument a server. Refer to the following examples.
- ▶ Restart the application server. Changes will take effect when you stop and then restart the server.

Repeat the instrumentation steps for every server on the machine involved in any data collection for the applications you will be profiling (usually, there will be only one application server, but it is possible for you to have more than one on a machine).

Examples

Consider the following examples:

- ▶ On a Linux machine, to instrument an IBM WebSphere Application Server V5.x server named server1, installed in the directory /opt/WebSphere/AppServer (no security):

```
./instrumentServer.sh -install -type IBM -serverName server1  
-serverHome /opt/WebSphere/AppServer -serverVersion 5
```

- ▶ On a Linux machine, to instrument a WebSphere Application Server V6.0 server named server2, installed in the directory /opt/WebSphere/AppServer, with profile name default and security enabled:

```
./instrumentServer.sh -install -type IBM -serverName server2
                     -serverHome /opt/WebSphere/AppServer -serverVersion 6
                     -profileName default -user <userId> -password <password>
```

- ▶ On a Linux machine, to instrument a BEA WebLogic application server (with specifics as indicated):

```
./instrumentServer.sh -install -type BEA -serverName server1
                     -serverHome /opt/bea/weblogic81 -javaHome /opt/bea/jdk141_02
                     -adminServerHost hostname.xyz.com -adminServerPort 7001
                     -user <userId> -password <password>
                     -startScript /opt/bea/weblogic81/mydomain/startManagedWeblogic.sh
```

- ▶ On a Windows machine, to instrument a WebSphere Application Server V5.x server named my_Server, installed in C:\Program Files\was5.x, with security enabled:

```
instrumentServer -install -type IBM -serverName my_Server
                 -serverHome "C:\Program Files\was5.x" -user <userId>
                 -password <password> -serverVersion 5
```

- ▶ On a Windows machine, to instrument a WebSphere Application Server V6.0 server named my_Server2, installed in C:\Program Files\was6.0, with security enabled, and profile name default:

```
instrumentServer -install -type IBM -serverName my_Server2
                 -serverHome "C:\Program Files\was6.0" -user <userId>
                 -password <password> -serverVersion 6 -profileName default
```

- ▶ On a Windows machine, to instrument a BEA WebLogic application server (with specifics as indicated):

```
instrumentServer -install -type BEA -serverName server1
                 -serverHome C:\bea\weblogic81 -javaHome C:\bea\jdk141_02
                 -adminServerHost localhost -adminServerPort 7001 -user <userId>
                 -password <password>
                 -startScript C:\bea\weblogic81\mydomain\startManagedWeblogic.cmd
```

Note: The WebLogic server has to be started with the JVM that is included with the product itself. Also, note that the JRockit VM is not a supported JVM. For managed WebLogic servers, the Java Home variable (under *Configuration → Remote Start*) must point to the Sun JVM shipped in WebLogic for an instrumented server to start correctly.

Instrumenting an application server using a GUI

The graphical user interface-based instrumenter enables you to instrument application servers on multiple local or remote hosts from your workstation (Figure 9-5). To use this tool, there are two requirements:

- ▶ You must first ensure that the Rational Performance Tester Agent V7.0 is installed on each remote machine.
- ▶ You must also have an SSH server installed on each machine (Linux SSH server or cygwin for Windows machines).

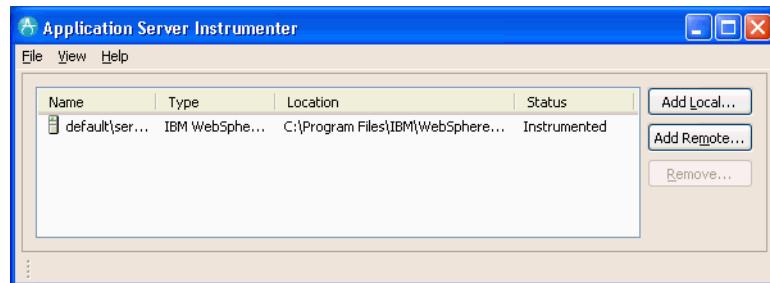


Figure 9-5 IBM Application Server Instrumenter graphical user interface

To invoke the graphical user interface based instrumenter, use the *Start* menu (Figure 9-6):

Start → *Programs* → *IBM Software Development Platform* → *IBM Rational Data Collection Infrastructure* → *Application Server Instrumenter*

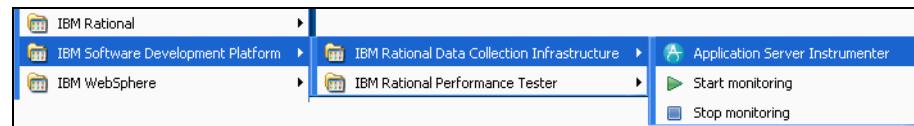


Figure 9-6 Start menu for IBM Software Development Platform

To instrument a local or remote application server, select *Add Local* or *Add Remote*, respectively. Either selection will prompt the user with a list of application servers to select.

For IBM WebSphere Application Server, the instrumentation parameters are shown in Figure 9-7.

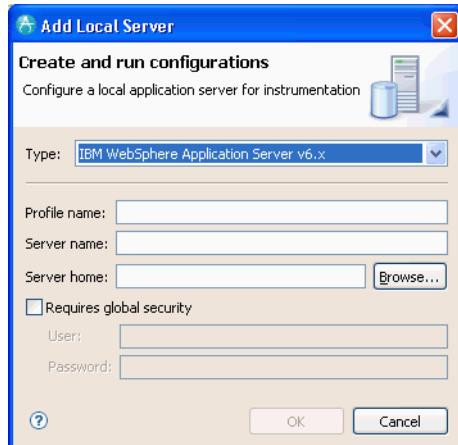


Figure 9-7 IBM WebSphere Application Server instrumentation parameters

- ▶ Profile name—The name of the profile that is set up on the server (only required for version 6).
- ▶ Server name—Server instance name (for example, server1).
- ▶ Server home—Server install home directory (for example, C:\Program Files\IBM\WebSphere\AppServer).
- ▶ Requires global security—Select *Requires global security* if the server requires authentication.
- ▶ User—The user ID used to authenticate with the server. This field is only activated if *Requires global security* is selected.
- ▶ Password—The password used to authenticate with the server. This field is only activated if *Requires global security* is selected.

For BEA WebLogic Application Server, the instrumentation parameters are shown in Figure 9-8.

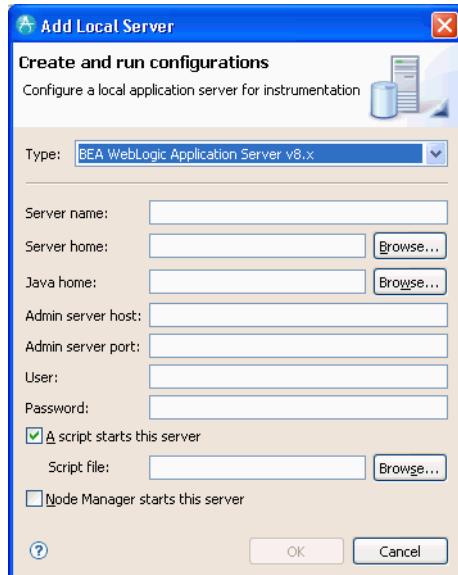


Figure 9-8 BEA WebLogic instrumentation parameters

- ▶ Server name—The name of the server (for example, server1).
- ▶ Server home—Server install home directory (for example, c:\BEA\WebLogic81).
- ▶ Java Home—Directory containing a Java Runtime Environment.
- ▶ Admin Server Host name—Domain administrative server host.
- ▶ Admin Server Port—Domain administrative server port.
- ▶ User—The user ID used to authenticate with the server.
- ▶ Password—The password used to authenticate with the server.
- ▶ A script starts this server —Full path to the script that starts the server.
- ▶ Node Manager starts this server—Used when the server is node managed.

When instrumenting a remote application server, you are presented with connection and advanced configuration options. The connection configuration is used to indicate the remote machine's SSH server configuration, such that a connection can be established to the machine (Figure 9-9).

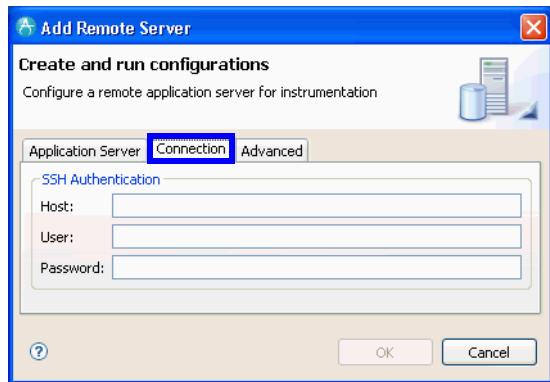


Figure 9-9 Remote instrumentation connection parameters

- ▶ Host—The host name of the server to instrument (for example, abc.ibm.com).
- ▶ User—The user ID used to authenticate with the SSH server.
- ▶ Password—The password used to authenticate with the SSH server.

The advanced configuration is used to indicate options for more tightly secure or customized SSH server configurations (Figure 9-10).

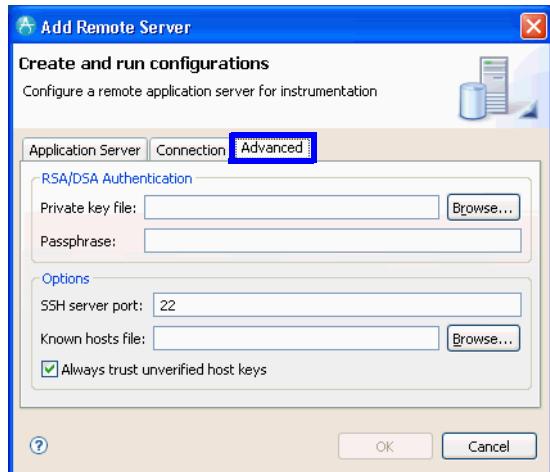


Figure 9-10 Remote instrumentation advanced configuration

- ▶ Private key file—The file that contains the private key data used to authenticate with the remote host. This field is optional.
- ▶ Passphrase—The passphrase used to authenticate with the private key file. This field is optional.
- ▶ SSH server port—The port of the remote SSH server.
- ▶ Known hosts file—The file that contains a list of known hosts. This field is optional.
- ▶ Always trust unverified host keys —Select this check box to force the utility to trust the unverified host key of the remote server.

If additional assistance is required when using the Application Server Instrumenter, there are two methods of help:

- ▶ The Help menu can display the tool's Help Contents (select *Help* → *Help Contents*) as shown in Figure 9-11.

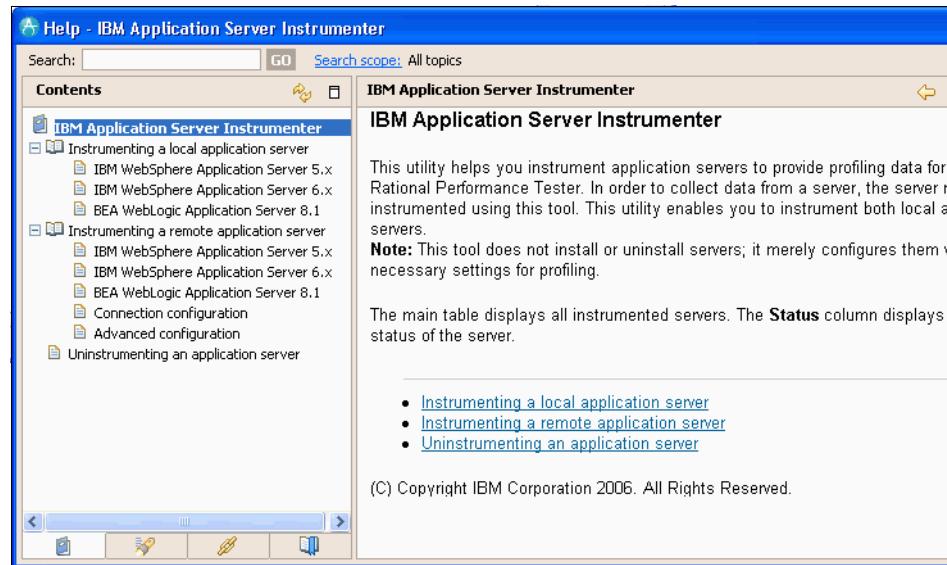


Figure 9-11 Application Server Instrumenter Help Contents

- ▶ Another option is to use context-sensitive help. To make use of this function, navigate to a dialog or input field. Then select it using the mouse, which will put the selected element into focus. Pressing F1 on the keyboard or the help icon? at the bottom-left side of a dialog displays help for the selected dialog or field (Figure 9-12).

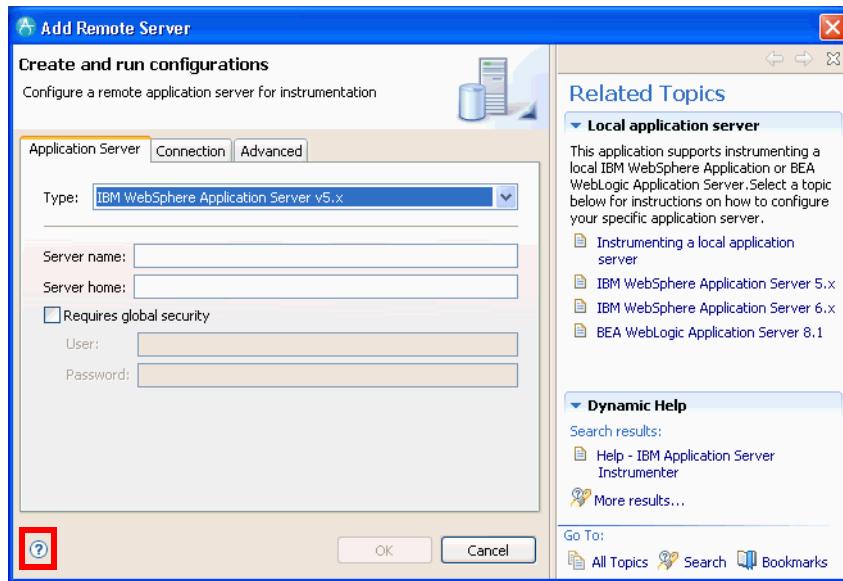


Figure 9-12 Application Server Instrumenter context-sensitive help (F1)

Examples

To instrument an IBM WebSphere Application Server V6.0 server named `server2`, installed in the directory `/opt/WebSphere/AppServer`, with profile name `default`, and security enabled, on a remote Linux machine with hostname `linux2`:

- ▶ Click *Add Remote*.
- ▶ Populate the Application Server tab as follows:
 - Type: select *IBM WebSphere Application Server v6.x*.
 - Profile name: `default`
 - Server name: `server2`
 - Server home: `/opt/WebSphere/AppServer`
 - Select *Requires global security*
 - User: `my_WAS_userID`
 - Password: `my_WAS_password`
 - Select or clear *Save password* as desired
- ▶ Populate the Connection tab as follows:
 - Host: `linux2`
 - User: `SSH_linux2_userId`
 - Password: `SSH_linux2_password`

- ▶ Populate the Advanced tab as follows:
 - Enter RSA/DSA authentication information
 - Enter desired options
- ▶ Click *OK*.

To instrument a BEA WebLogic application server with server named server1, installed in the directory /opt/bea/weblogic81, Java home /opt/bea/jdk15, admin server host hostname.xyz.com, admin server port 7001, and start script file /opt/bea/weblogic81/mydomain/startManagedWeblogic.sh, on a remote Linux machine with hostname linux3:

- ▶ Start the WebLogic server.
- ▶ Click *Add Remote*.
- ▶ Populate the Application Server tab as follows:
 - Type: select *BEA WebLogic Application Server v8.x*
 - Server name: server1
 - Server home: /opt/bea/weblogic81
 - Java home: /opt/bea/jdk15
 - Admin server host: hostname.xyz.com
 - Admin server port: 7001
 - User: my_BEAL_userId
 - Password: my_BEAL_password
 - Select *A script starts this server*
 - Script file: /opt/bea/weblogic81/mydomain/startManagedWeblogic.sh
- ▶ Populate the Connection tab as follows:
 - Host: linux3
 - User: SSH_linux3_userId
 - Password: SSH_linux3_password
- ▶ Click *OK*.
- ▶ Stop and restart the server.

Uninstrumenting an application server from the command line

To use the command line to uninstrument an application server, follow these steps:

- ▶ Open a command line and go to the rpa_prod directory under the data collection infrastructure installation directory.
- ▶ The instrumentation utility, which is also used to uninstrument servers, is called `instrumentServer.sh` (`instrumentServer.bat` on Windows). Type the command name with no arguments to see the syntax details for the command.

- ▶ Type the command name with the `-uninstall` argument and all of the other arguments you used to instrument it originally. For example, on Windows, to uninstall an IBM WebSphere Application Server V5.1 server instance named `my_Server`, installed in `C:\Program Files\was5.1`, with security enabled:

```
instrumentServer -uninstall -type IBM -serverName my_Server
    -serverHome "C:\Program Files\was5.1" -user my_WAS_userId
    -password my_WAS_password -serverVersion 5
```

- ▶ Restart the server.

Note: If you have uninstalled the server or removed the server instance without uninstrumenting it, the `instrumentServer` utility still thinks that the server is there, but will be unable to contact the server to uninstrument it. This will block the uninstallation process for the data collection infrastructure.

Repeat the uninstrumentation steps for every server instrumented for data collection. When you are finished, the `InstrumentationRegistry.xml` file will be empty, and data collection uninstallation will proceed.

Uninstrumenting an application server using the GUI

To use the GUI to uninstrument an application server, follow these steps:

- ▶ Select the server you wish to uninstrument (see Figure 9-5 on page 271).
- ▶ Click *Remove*.
- ▶ Restart the server.

Repeat the uninstrumentation steps for every server instrumented for data collection. When you are finished, the `InstrumentationRegistry.xml` file will be empty, and data collection uninstallation will proceed.

The `InstrumentationRegistry.xml` file is a data file that persists meta data information about each instrumented server. On a Windows installation, the file is located at:

`C:\Program Files\IBM\SDP70\DCI\rpa_prod\tivoli_comp\app\instrument\61\bin`

The Performance Tester uninstall wizard also uses this file to determine if application servers are still instrumented at the time of uninstalling the product. If this scenario is true, then the wizard will halt and warn the user to use the ASI or command line tools to uninstrument the application server.

You cannot uninstall Performance Tester without uninstrumenting the application server because doing so would leave the server in an unusable state, as program files would be missing that are required by the instrumentation.

9.3 Enabling real-time application monitoring

Real-time application monitoring enables a Performance Tester to take a pre-emptive approach to isolating bottlenecks in applications during the development and test phases of the *Information Technology Lifecycle Management* (ITLM). This approach is advantageous because problems can be identified before the application is deployed into the production environment, and therefore, reducing the number of critical problems that could arise soon after the application is released for consumption by customers. Alternatively, real-time application monitoring can be used to validate the performance of an application after it has been patched by the development team.

9.3.1 Architecture

Real-time application monitoring in Performance Tester is enabled by utilizing the data collection infrastructure (DCI). The role of the DCI is to transform and forward the ARM events, which are reported to the Tivoli ARM engine by an ARM instrumented application, to the client (Figure 9-13).

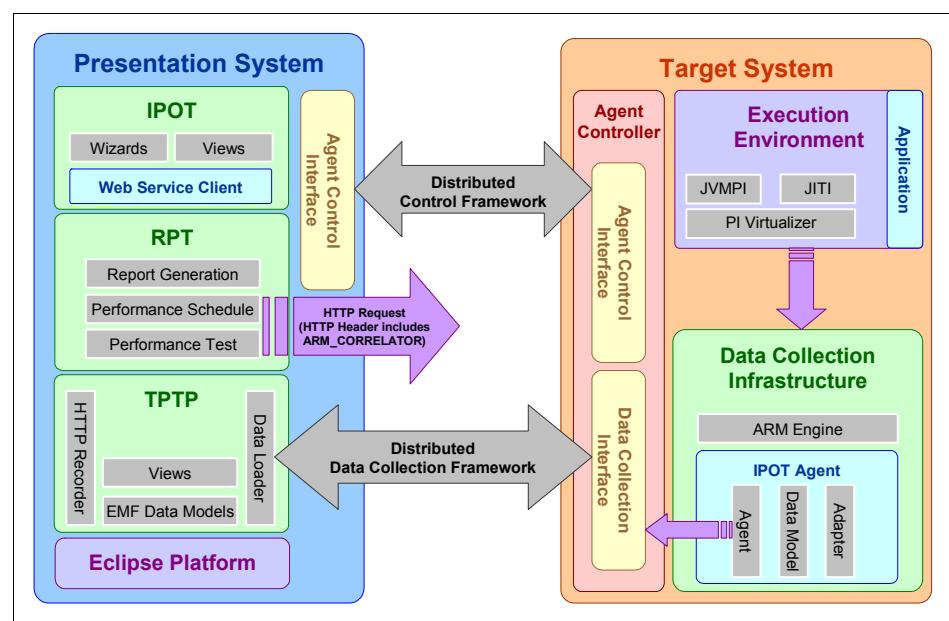


Figure 9-13 Environment system architecture using the Data Collection Infrastructure

Transaction flow

To understand the architecture, let us consider the flow of a business transaction through the environment:

- ▶ The DCI located on the machine where the root (or edge) transaction will be executed must be started for monitoring. For example, executing a HTTP request for `http://ibm.com/PlantsByWebSphere` indicates that the application `PlantsByWebSphere` has to be instrumented and that a DCI is installed on the machine located at `ibm.com`.
- ▶ A client invokes an HTTP transaction involving an application that is ARM instrumented. Two clients can be used:
 - Performance Tester—Behaves similar to a browser by executing HTTP requests for each page in the test. When response time breakdown (RTB) is enabled, the Performance Tester client adds the `ARM_CORRELATOR` header attribute to the request, which enables the DCI to monitor the transaction. This client automatically establishes a connection to the DCI on the machine where the root transaction will be executed.
 - Internet browser—Executing HTTP requests for a selected URL. The user must manually establish a connection to the DCI on the machine where the root transaction will be executed.
- ▶ As the transaction invokes the application operating in the execution environment, the instrumentation code (or probes) will execute. These probes initiate ARM transactions that are monitored by the Tivoli ARM engine.
- ▶ The IPOT agent is a Java-based application that implements the ARM plug-in interface provided by Tivoli for registering third-party applications with the Tivoli ARM engine. As ARM events are created and reported to the engine, they are also reported to the IPOT agent. The events are collected and organized into a transactional hierarchy, from the root transaction to include all of its sub-transactions. This hierarchy is then converted into a set of events that are modeled after the TPTP trace model format. The set of events is then sent to the presentation system.
- ▶ The target and presentation systems communicate with each other using the Agent Controller. This component integrates with Eclipse TPTP, which Performance Tester is based on. The component also handles managing (starting and stopping monitoring) of the IPOT Agent.

Each ARM event reported to the DCI contains information about the start time and completion time of the ARM transaction, in addition to meta data. The timing information helps to compute metrics that help an analyst determine if a performance problem is present. The meta data helps to indicate the context of the transaction in the entire transaction hierarchy and the type of transaction being invoked.

Execution environment

Within the execution environment, there are two agents that implement the Java profiling interface, JITI and JVMPPI. JITI has already been discussed earlier. The JVMPPI agent (or Java profiling agent) used in Eclipse TPTP has been enhanced in Performance Tester to include features such as security and support for Citrix and SAP integration.

When you set up a profiling configuration to monitor an application, you choose an analysis type to specify the type of data that you want to collect. Depending on the analysis type that you choose, the data collection infrastructure uses one or both of two agents to collect the data. The agent used is selected automatically to correspond to your profiling settings.

The following two collection agents can be used:

- ▶ The ARM agent is most useful, and is selected automatically, in the following scenarios:
 - When analyzing J2EE application performance and application failures, especially in distributed applications where event correlation is essential.
 - When profiling ARM-instrumented applications. For example, if you ARM-enable a database, you could view a transaction go from your application into the database, see all the processing in the database, and then see the response return, all displayed in one sequence diagram.
- ▶ The JVMPPI agent is most useful, and is selected automatically, in the following scenarios:
 - When performing memory analysis (for example, leak analysis).
 - When examining object interactions (for example, for the UML2 Object Interactions view). In most situations, class interaction analysis (available with the ARM agent) is enough to help find problems, but occasionally you will need to do further profiling at the object level.
 - When profiling non-J2EE Java applications, such as J2SE™ or J2ME™ applications.

Each agent offers its own set of data collection features, as shown in Table 9-1.

Table 9-1 Comparison of data collection features between ARM and JVMPPI agents

| Feature | ARM agent | JVMPPI agent |
|--|--------------|--------------|
| Provides the ability to correlate remote calls between processes and hosts | Yes | No |
| Affects application performance when the agent runs in profiling mode | Small effect | Large effect |

| Feature | ARM agent | JVMPI agent |
|---|--|-------------------------------|
| Offers filtering mechanisms | By J2EE component type (for example servlet or JDBC), host URL | By package, class, and method |
| Collects memory information and object interactions (for the UML2 Object Interactions view) | No | Yes |

With two agents that can potentially be used in parallel, the load on the execution environment can increase dramatically. As a result, the profiling interface (PI) virtualizer (virt.dll on Windows systems) was added. This component is the only agent that the JVM is aware of from its' perspective. When the PI virtualizer received events from the JVM, it broadcasts those same events to every JVMPI-based agent it is aware of. In this case, those agents are JITI and the Java profiling agent. To use the Java profiling agent, one would add the following VM argument:

```
-XrunpiAgent:server=enabled
```

When the PI virtualizer is present, the following VM argument is used:

```
-Xrunvirt:agent=jvmPI:C:\PROGRA~1\IBM\DCI\rpa_prod\TIVOLI~1\app\instrument
\61\appServers\server1_100\config\jiti.properties,agent=piAgent
:server=enabled
```

Notice that the argument specifies both agents to which to broadcast the events. This configuration can be found in the same configuration files as the instrumentation, as previously discussed.

Data collection infrastructure

To collect end-to-end transaction information, it is required to install the DCI on all systems involved in the path of a transaction (Figure 9-14). This requirement is present because the ARM correlator is flown across the physical machine boundary. As a result, when a ARM correlator is sent from one machine to another, the DCI on the machine receiving the ARM correlator has to execute special processing known as *dynamic discovery*.

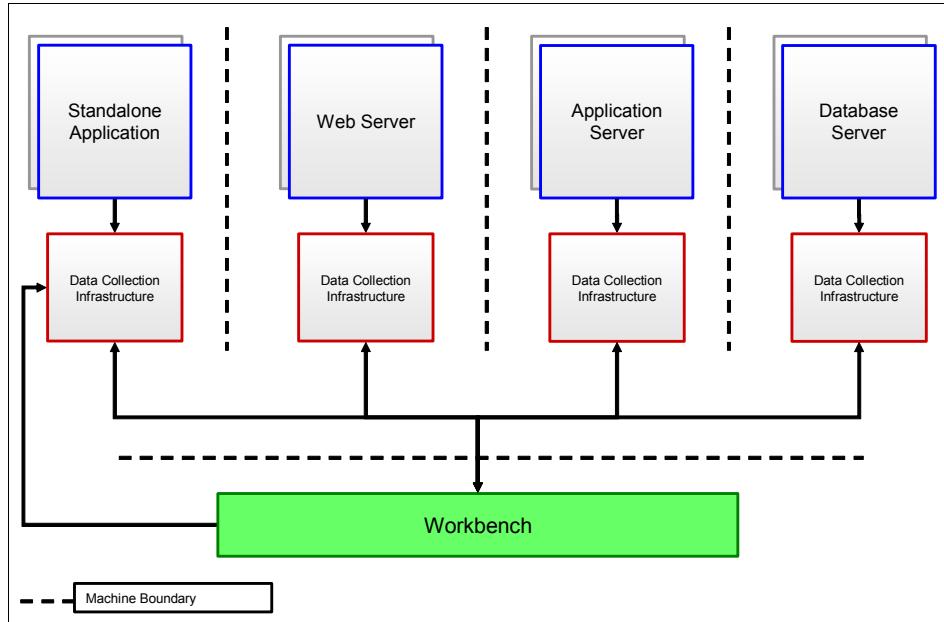


Figure 9-14 Data collection infrastructure for a distributed environment

The purpose of the dynamic discovery process is to automatically have the client workbench attach and begin monitoring the DCI located on the machine where the remote method is being invoked. This is required because when a transaction is first executed, the client is only attached to the first machine in the transactions' path. Therefore, rather than have the user know all the physical machines that would be involved in any given transaction and manually attach to each, this process is automated.

This process is made possible by sending information in the ARM correlator about the Agent Controller on the caller machine. Therefore, when the RMI-IIOP transaction is invoked, an ARM event is sent to the DCI at this newly discovered machine. Specifically, at the IPOT agent, this RMI-IIOP transaction is detected. At this moment, the IPOT agent invokes a peer request to the Agent Controller at the caller machine, using the Agent Controller at the callee machine. This request asks for the known client attached at the caller machine to also establish a connection to the newly discovered machine (Figure 9-15). In turn, transactional information from the callee machine flows to the client for analysis.

Note: The security feature on the Agent Controller must be disabled for dynamic discovery to work. The security feature is not yet fully functional across distributed environments. To disable or verify that it is disabled, execute the SetConfig script located in the bin directory of the Agent Controller installation.

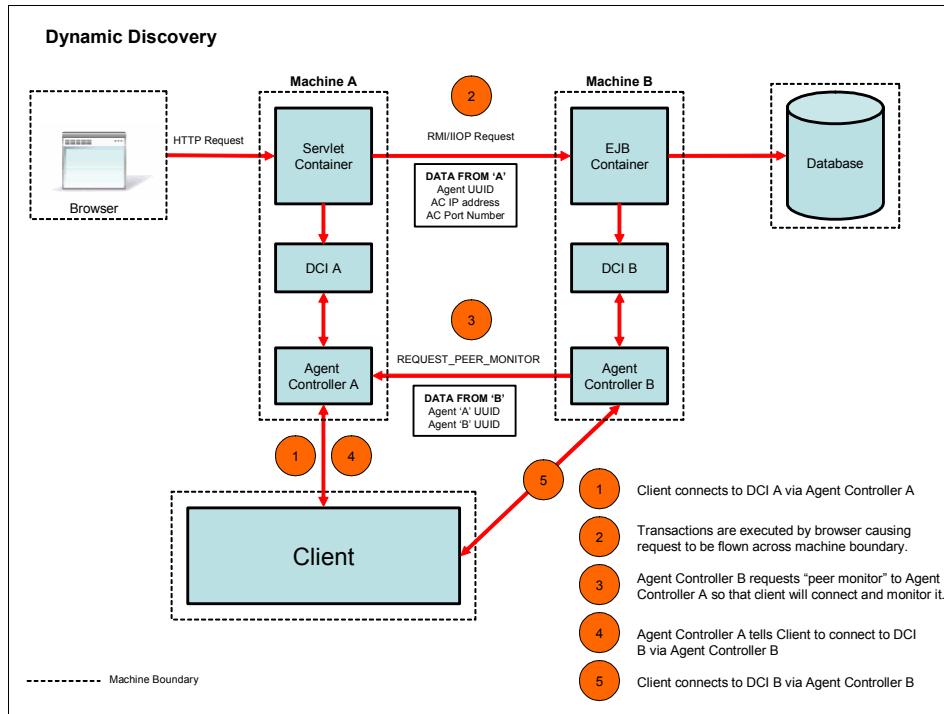


Figure 9-15 Dynamic discovery process

For the most part, the focus has been on collecting transactional information from application servers. However, there are a few database systems, such as IBM DB2, which have native ARM instrumentation supported in the product. Enabling this ARM instrumentation allows deeper monitoring of end-to-end transaction monitoring.

For example, rather than just being able to know that a transaction has used JDBC to invoke an SQL transaction from a Java application, analysts can have the transaction information from within the database for that specific query. This information greatly helps to narrow down if a solution problem is because of an Java application or the solution's database.

Following the database product's instructions to enable this ARM instrumentation is the first step to enable end-to-end transaction monitoring within the database. The second step is to install the DCI on the same machine where the database executes and then starting the DCI in the monitoring mode will allow database transaction information to be sent to the client.

Furthermore, the environment can be configured to collect the exact SQL query being executed on a database. To enable this configuration, you must disable database privacy and security for the local DCI. The instructions for enabling the collection of SQL statements are as follows (recommended to be done after you have already instrumented the application server):

- ▶ Shut down the application server and DCI.
- ▶ Navigate to the following directory:
`<DCI_INSTALL_DIRECTORY>/rpa_prod/tivoli_comp/app/instrument/61/appServers /<servername>/config`
- ▶ Open the file monitoringApplication.properties file.
- ▶ Add the following two lines:
`tmtcp.isPrivacyEnabled=false
tmtcp.isSecurityEnabled=false`
- ▶ Start Monitoring for the DCI.
- ▶ Start the application server.
- ▶ Initiate data collection by invoking transactions.

9.3.2 Configuring Performance Tester

You can use response time breakdown to see statistics on any page element captured while running a performance test or performance schedule.

Response time breakdown shows how much time was spent in each part of the system under test as the system was exercised. The response time breakdown view is associated with a page element (URL) from a particular execution of a test or schedule. This shows the *insides* of the system under test, because the data collection mechanisms are on the systems under test, not the load drivers.

Typically you will capture response time breakdown in real-time in development, or test, environments, rather than production environments. To capture response time breakdown data, you must enable it in a test or schedule, and configure the amount of data to be captured.

To configure a performance test, a check box exists in the Test Element Details section of the Performance Test Editor (Figure 9-16). If the top-most node in the Test Contents tree is selected—which is the performance test itself—then selecting *Enable response time breakdown* from the Test Element Details causes application monitoring on every page and page element in the performance test. If only a specific page or page element requires application monitoring, then select it from the Test Contents tree. This action displays the selected items' configuration in the Test Element Details pane, from which you can enable response time breakdown.

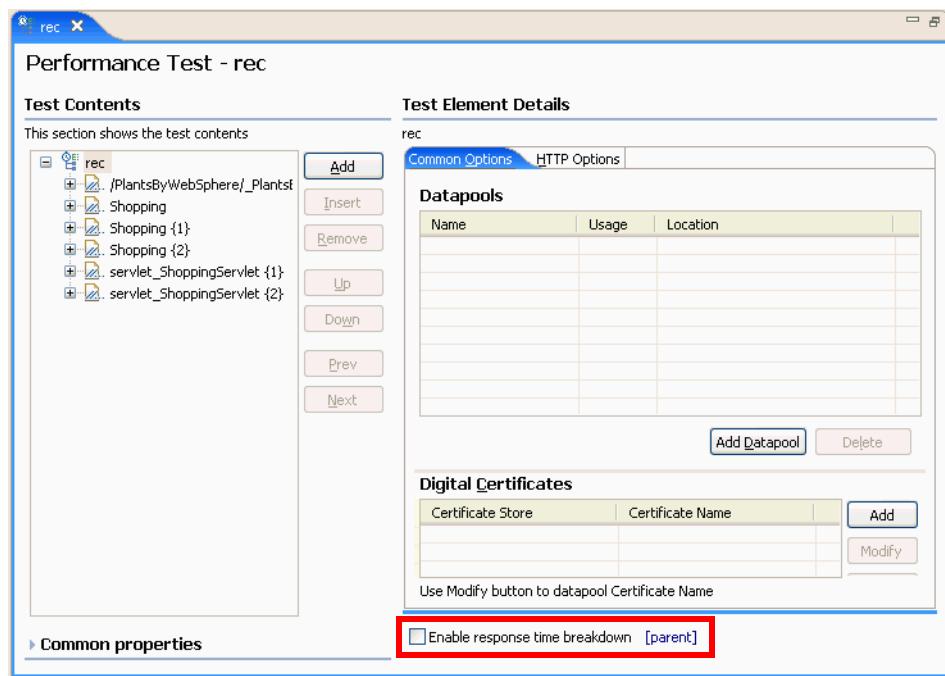


Figure 9-16 Response time breakdown configuration in a performance test

To configure a performance schedule, you can enable response time breakdown from the Performance Schedule Editor (Figure 9-17). Under the Schedule Element Details pane, select the *Response Time Breakdown* tab and select *Enable collection of response time data*. This activates the test list and Options. After having enabled response time breakdown data collection, you have to set logging detail levels.

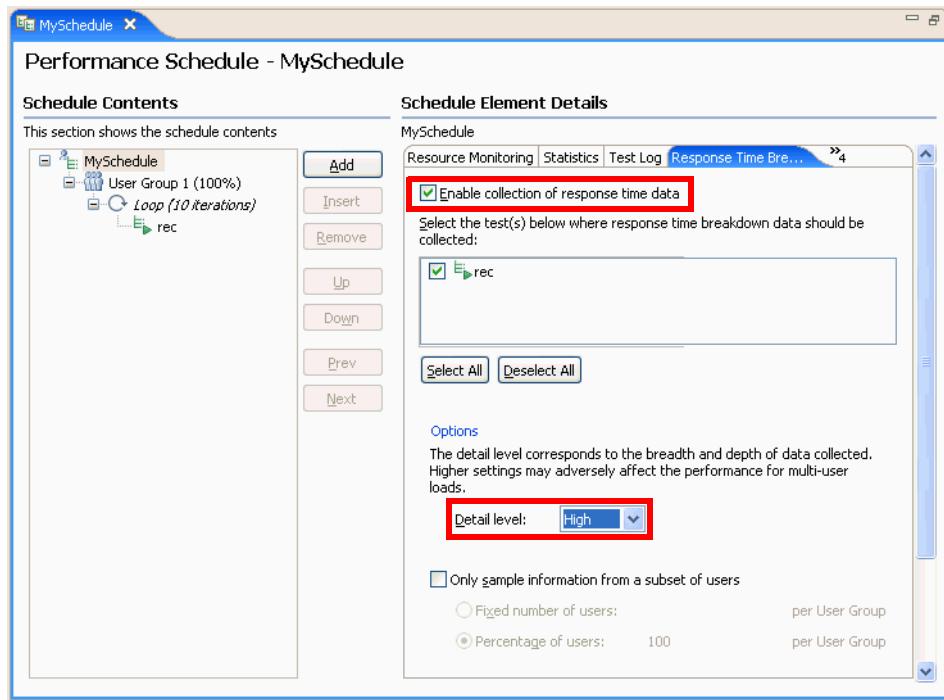


Figure 9-17 Response time breakdown configuration in a performance schedule

The Response Time Breakdown page allows you to set the type of data that you see during a run, the sampling rate for that data, and whether data is collected from all users or a representative sample.

- ▶ Enable collection of response time data—Select this option to activate response time breakdown collection. This shows you the response time breakdown for each page element.
- ▶ Detail level—Select *Low* or *Medium* to limit the amount of data collected. The higher the detail level, the deeper the transaction is traced and the more context data is collected. For example, at *Low*, only surface-level transactions are monitored. In a J2EE-based application, this includes Servlets. As the detail level is increased, collection for EJBs and RMI aspects of a J2EE application are collected.
- ▶ Only sample information from a subset of users—if you set the detail level to *High* or *Medium*, set a sampling rate to prevent the log from getting too large.
- ▶ Fixed number of users—the number that you select is sampled from each user group. Unless you have specific reasons to collect data from multiple users, select *Fixed number of users* and specify one user per user group.

- ▶ Percentage of users—The percentage that you select is sampled from each user group, but at least one user is sampled from each user group.

Enabling response time breakdown in a performance test will not affect the response time breakdown in any performance schedule that is referencing it. In addition, enabling response time breakdown in a performance schedule will not affect the response time breakdown configuration in a performance test. Both, the test and schedule are separate entities.

After making the appropriate response time breakdown configuration, you must execute the test or schedule to initiate application monitoring during the load test. This action can be executed by using the *Run* menu to launch the test or schedule. Response time breakdown can only be exploited when executing a test or schedule from the graphical-based user interface. If a test or schedule is configured for response time breakdown and then executed using the command line mechanism, response time breakdown data will not be collected.

9.3.3 Profiling for real-time observation

Within a development environment, you can collect response time breakdown data for analysis using several methods:

- ▶ You can profile (collect response time breakdown data from) a running distributed J2EE application in a development environment.
- ▶ You can also profile applications as they are exercised by automated testing tools, which free you from having to repeatedly run the problem scenario, and which can simulate load on the application.
- ▶ You can profile a Web service component of an application using IBM WebSphere Application Server.
- ▶ You can profile non-J2EE applications, or applications in any other language supported by the ARM standard.

To collect real-time response time breakdown data, make sure that you comply with the following requirements:

- ▶ The data collection infrastructure must be installed, configured, and running on all computers from which data is to be collected. See the installation guide for more information.
- ▶ The Agent Controller (part of the data collection infrastructure) port on all involved computers must be set to the default (10002).
- ▶ The application system must not involve communication between networks that use internal IP addresses and network address translation.

To profile or to collect real-time response time breakdown data from an application system, you must first establish a connection to the data collection infrastructure. First identify the server that will process the initial transaction request. More specifically, identify the first application server that the transaction request reaches.

Note: You do not have to explicitly create a connection to every computer involved in the distributed application that you want to monitor. Through a process called dynamic discovery, the workbench automatically connects to each computer as the transaction flows to it.

To initiate real-time data collection, select *Run* → *Profile* or use the toolbar button to open the launch configuration dialog (Figure 9-18). This profile action asks you to switch into the Profile and Logging perspective of the Performance Tester product. This perspective enables profiling tools and has a different layout of the views on the workbench.



Figure 9-18 Profile launch configuration action

The profile configuration dialog allows you to create, manage, and run configurations for real-time application monitoring. This method for monitoring is used for J2EE, non-J2EE, and Web service applications that do not have automated application monitoring—as is the situation with a performance test or schedule.

- ▶ In the Launch Configuration dialog, select *J2EE Application* and click *New*. If your application is not running on a J2EE application server, but rather is ARM instrumented manually, select *ARM Instrumented Application* instead.
- ▶ On the **Host** page (Figure 9-19), select the host where the Agent Controller. If the host you need is not on the list, click *Add* and provide the host name and port number. Test the connection before proceeding.

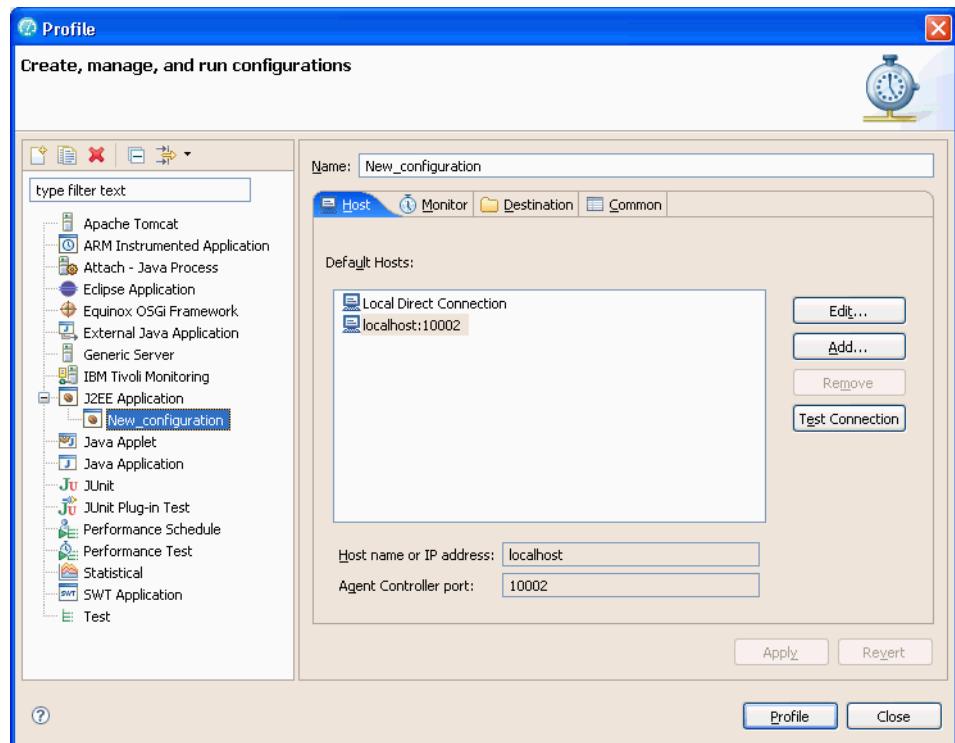


Figure 9-19 Profile launch configuration for J2EE application: Host page

- ▶ On the **Monitor** page (Figure 9-20), select the J2EE Performance Analysis analysis type—or the ARM Performance Analysis analysis type if profiling using the ARM Instrumented Application launch configuration. If you want to customize the profiling settings and filters, click *Edit Options*.

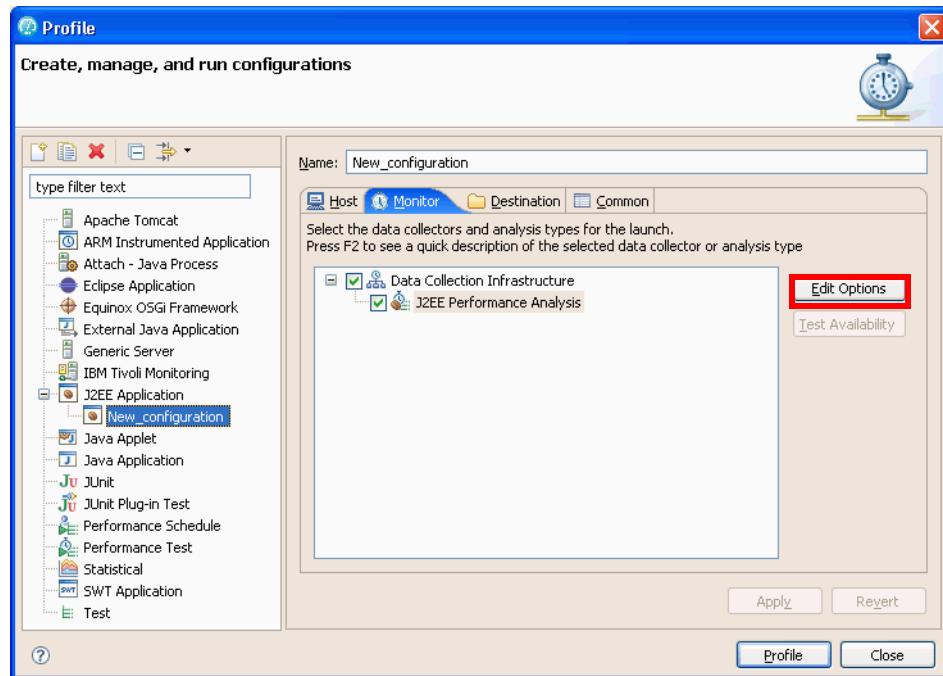


Figure 9-20 Profile launch configuration for J2EE application: Monitor page

- On the **Components** page (Figure 9-21), select the types of J2EE components from which you want to collect data.

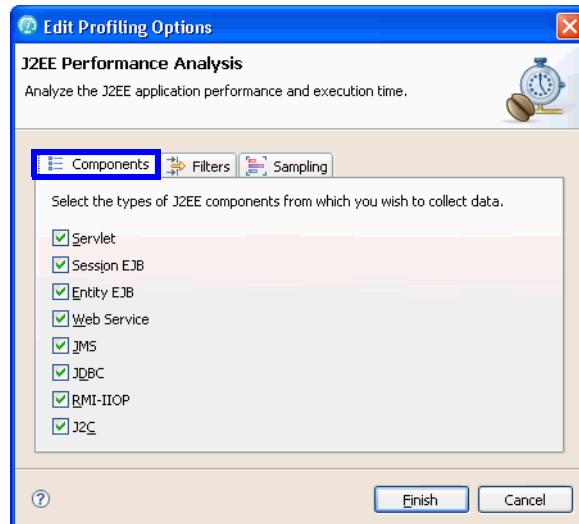


Figure 9-21 J2EE performance analysis components to monitor

- On the **Filters** page (Figure 9-22), specify the hosts and transactions that you want to monitor. The filters indicate the types of data that you do want to collect, that is, they include, rather than exclude.

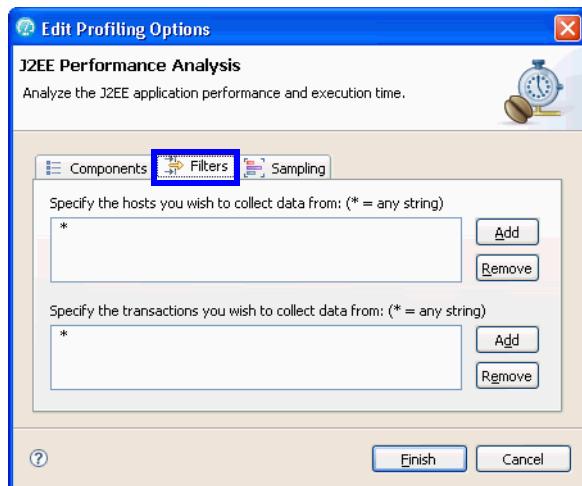


Figure 9-22 J2EE performance analysis filters to apply

- On the **Sampling** page (Figure 9-23), you can limit the amount of data being collected by specifying a fixed percentage or a fixed rate of all the data to collect.

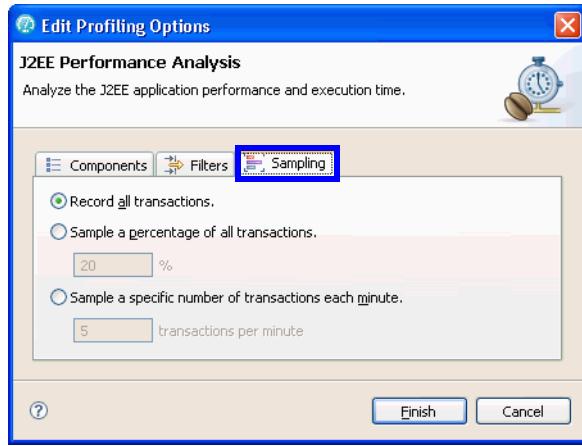


Figure 9-23 J2EE performance analysis sampling options

- Sample a percentage of all transactions: The profiler alternates between collecting and discarding calls. For example, with a setting of 25%, the first call is collected; the next three are not.

- Sample a specific number of transactions each minute: The first n calls (where n is the specified value) are collected, and nothing else will be collected that minute. Thus, you will never receive data for more than n calls in any given minute.
- ▶ You can set up filters and sampling to specify how much data is collected. A filter specifies which transactions you want to collect data from. Sampling specifies what subset percentage or number of transactions you want to collect data from.

Filters and sampling work at the root (or edge) transaction level. A root, or edge, transaction is one that is not a sub-transaction of any other (at least, as far as the data collection is concerned).

Thus, when using a browser to access a Web application, a root transaction is a URL request when it first hits the server. The filtering and sampling apply at this level. That is, if you enter a transaction filter, it acts on the URL, and if you sample, it will collect only some of the URLs, and discard others. It is all or nothing; it does not sample parts of the URL transactions.

If the Web application resides on multiple servers, some instrumented and some not, only data about instrumented servers will be included. For example, if the user accesses the application through Server A, which is not instrumented, and Server A contacts through a method call Server B, which is instrumented, then the method call is the root transaction. Filtering will not work because it uses URLs, not method names, to filter.

Also, if you are using performance tests or schedules to generate profiling data, then the root transaction takes place on the first application response measurement (ARM) instrumented test element to be run. If the entire test or schedule is ARM-instrumented (that is, *Enable response time breakdown* is selected for the top-level test or schedule element), there will only be one root transaction, and so filtering and sampling will be ineffective.

Before you start monitoring, bring the application to the state right before the performance problem trigger. For example, if an action on a particular Web page is slow, navigate to that page.

- ▶ Click *Profile* (Figure 9-20 on page 291). The connected agent and its host and process will be shown in the Profiling Monitor view. Depending on your profiling configuration, more than one agent might be shown if operating in a distributed environment.
- ▶ Start the monitoring agent by selecting the agent and *Start Monitoring* (Figure 9-24). If there is more than one agent for this profile configuration, start monitoring each of the agents. Any activity in the application that fits the profiling settings that you specified earlier will now be recorded. The agent state will change from <attached> to <monitoring>, and to <monitoring...collecting> whenever data is being received from the agent.

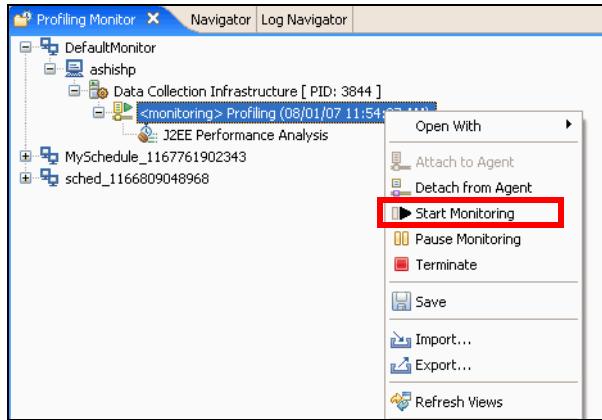


Figure 9-24 Profiling Monitor for J2EE Performance Analysis

- ▶ In the application, perform the steps required to trigger the performance problem.
- ▶ Stop monitoring the agent by selecting *Stop Monitoring* from the context menu. For best results, detach from the agent so that other users can profile the system. Select the agent and *Detach* in its context menu. Repeat this step for each agent involved in collecting the data.

9.3.4 Finding the causes of performance problems

Application failures can be caused by a number of coding problems, each of which you must investigate using the data collection and analysis techniques and tools available. Typical application failures include stoppages, when the application unexpectedly terminates, and lockups, when the application becomes unresponsive, for example, as it enters an infinite loop or waits for an event that will never happen.

In the case of a lockup, you might not see any actual errors logged. Use interaction diagrams, statistical views, and thread analysis tools to find the problem. For example, if the problem is an endless loop, the UML sequence diagrams show you repeating sequences of calls, and statistical tables show methods that take a long time. If the problem is a deadlock, thread analysis tools will show that threads you expect to be working are actually waiting for something that is never going to happen.

After you have collected response time breakdown data, you can analyze the results in the profiling tools to identify exactly what part of the code is causing the problem.

Generally, you first narrow down the component (which application on which server) has the problem. Then, you can continue narrowing down to determine which package, class, and finally which method is causing the problem. When you know which method the problem is in, you can go directly to the source code to fix it.

You can view response time breakdown data in the Test perspective or in the Profiling and Logging perspective if a performance test or performance schedule is executed with response time breakdown; otherwise, if you manually launched the J2EE application or ARM Instrumented application launch configuration, then the data collected is only viewable in the Profile and Logging perspective.

Tracking such problems can involve trial-and-error as you collect and analyze data in various ways. You might find your own way that works best.

Performance Tester report analysis

After a performance test or performance schedule has completed execution, the default Performance Tester report will be displayed to the user (Figure 9-25).

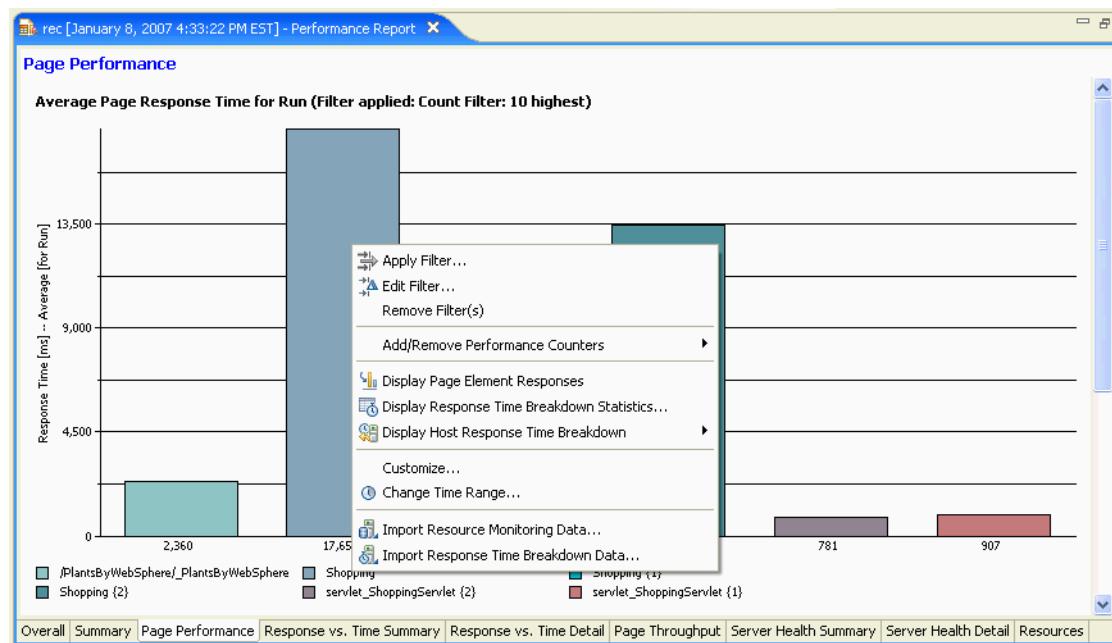


Figure 9-25 Page Performance report

There are two mechanisms that can be used to analyze response time breakdown data:

- ▶ Response Time Breakdown Statistics—A tabular format of a transaction and its' sub-transaction response time for a particular page or page element.
- ▶ Interactive Reports—A graphical drill-down process for decomposing a transaction.

In these two reporting mechanisms a hierarchy exists, allowing a means of organizing and categorizing a transaction and its sub-transactions. The hierarchy is as follows:

- ▶ Host—The physical machine where the application is executing the method invocation, in context.
- ▶ Application—The containing application where the method invocation was observed. Typically, for J2EE applications, this is an application server.
- ▶ Application Component—A categorization label, or meta data, give to the method invocation, in context. For J2EE applications, a method invocation might be labeled as *Session EJB™*, for which this label is also the application component.
- ▶ Package—The package where the class and method invocation, in context, belong to.
- ▶ Class—The class where the method invocation, in context, belongs to.
- ▶ Method—An invocation of a specific method.

Analysis using the Statistics view

Selecting the *Display Response Time Breakdown Statistics* option from the Page Performance report displays a tabular format of a transaction and its sub-transaction response time for a particular page or page element. First a Page Element Selection Wizard (Figure 9-26) is presented so you can pick a particular page element to view its transaction decomposition. All page elements are listed in descending order from the amount of response time for each element.

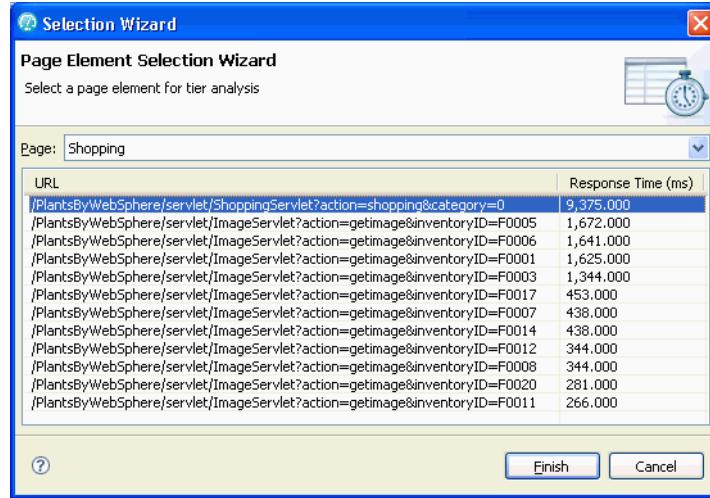


Figure 9-26 Page Element Selection Wizard

The Response Time Breakdown Statistics view displays an aggregation all the sub-transactions for the selected page element, which is listed on the top of the view (Figure 9-27).

| Page Performance > Response Time Breakdown Statistics | | | | | | |
|---|--------------------------|-------------------|--------------------|------------------|-------------------|-------|
| Method | Class | Package | Base Time (sec...) | Average Base ... | Cumulative Tim... | Calls |
| localhost:9080/PlantsByWebSphere/servlet/ShoppingServlet?action=shopping&category=0 | RPT | default package | 9.620 | 9.620 | 9.620 | 1 |
| localhost:9080/PlantsByWebSphere/servlet/ShoppingServlet?action=shopping&category=0 | RPT | default package | 9.171 | 9.171 | 9.171 | 1 |
| * getItemsByCategory(int) | CatalogBean | com.ibm.websph... | 4.179 | 4.179 | 8.984 | 1 |
| getPkgInfo() | ConcreteInventory_530... | com.ibm.websph... | 0.453 | 0.023 | 0.453 | 20 |
| getID() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.373 | 0.019 | 0.689 | 20 |
| getName() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.348 | 0.017 | 0.511 | 20 |
| isPublic() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.340 | 0.017 | 0.638 | 20 |
| getPkgInfo() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.316 | 0.016 | 0.769 | 20 |
| getImage() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.314 | 0.016 | 0.352 | 20 |
| getID() | InventoryBean | com.ibm.websph... | 0.283 | 0.014 | 0.316 | 20 |
| getPrice() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.275 | 0.014 | 0.316 | 20 |
| getDescription() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.270 | 0.013 | 0.325 | 20 |
| isPublic() | InventoryBean | com.ibm.websph... | 0.235 | 0.012 | 0.298 | 20 |
| getHeading() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.215 | 0.011 | 0.229 | 20 |
| getNotes() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.214 | 0.011 | 0.273 | 20 |
| getCategory() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.207 | 0.010 | 0.240 | 20 |
| DeliveryTime | localhost:Plant | default package | 0.204 | 0.204 | 0.204 | 1 |
| getQuantity() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.176 | 0.009 | 0.268 | 20 |
| getCost() | EJSLocalCMPInventory_... | com.ibm.websph... | 0.167 | 0.008 | 0.191 | 20 |
| getName() | ConcreteInventory_530... | com.ibm.websph... | 0.163 | 0.008 | 0.163 | 20 |
| getQuantity() | ConcreteInventory_530... | com.ibm.websph... | 0.092 | 0.005 | 0.092 | 20 |
| isPublic() | ConcreteInventory_530... | com.ibm.websph... | 0.063 | 0.003 | 0.063 | 20 |

Figure 9-27 Response Time Breakdown Statistics view

There are various tools available in this view that can be helpful when analyzing a performance problem. Use the navigation information in the upper left corner to navigate back to previous views. Use the toolbar in the upper right corner to toggle between tree and simple layouts; add filters; select which columns are displayed; jump to source code; toggle between percentages and absolute values; and export the table to CSV, HTML, or XML format.

The tree layout shows the following hierarchy, in order: Host, application, component, package, class, method (Figure 9-28). Each host comprises a tier in your enterprise environment. Within each host, there are tiers of applications. Within each application, there are tiers of components, and so on. The tree layout helps you identify which tier has the slowest response time.

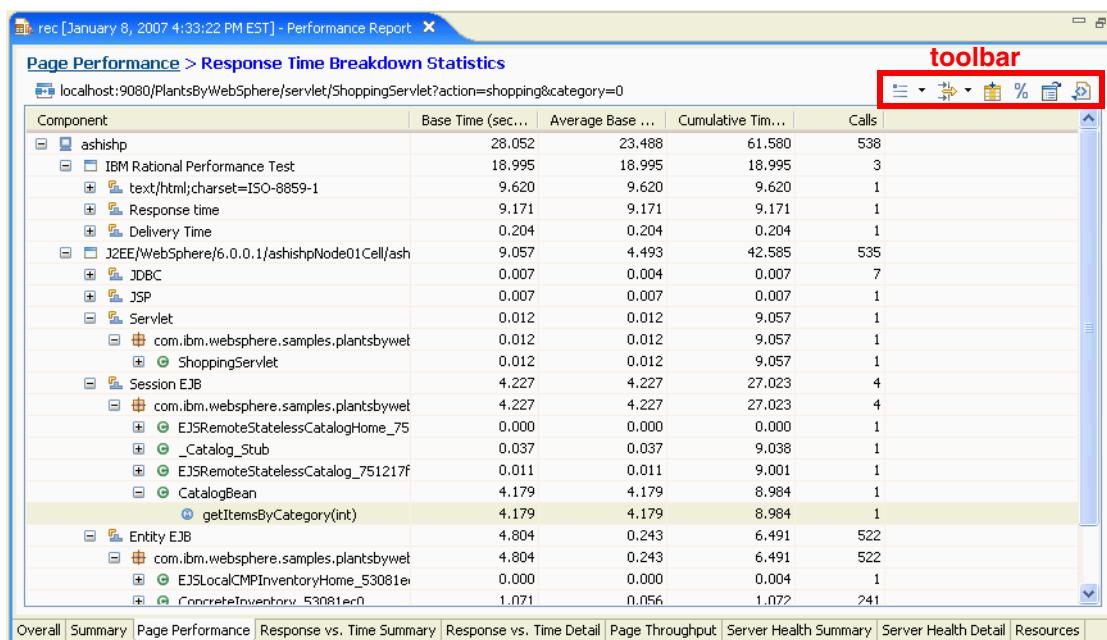


Figure 9-28 Response Time Breakdown Statistics: Tree layout

Use the first icon in the toolbar in the upper right corner to toggle between tree and simple layouts. The simple layout is a flattened version of the tree layout. Use this layout if you want to see all of the methods without seeing the relationships shown in the tree layout. The simple layout provides a quick and easy way to see the slowest or fastest methods.

The default layout is the simple layout. Click on a column heading to sort the table by that column. Drag the columns to change the order in which they are displayed. Except for the first column in the tree layout, all of the columns are moveable.

The exact URL of the selected page element is displayed above the table.

Four results (in seconds) are displayed for each object:

- ▶ Base Time—The time spent inside this object, excluding time spent in other objects that the selected object invokes.
- ▶ Average Base Time—The time spent inside this object, excluding time spent in other objects that the selected object invokes, divided by the number of calls.
- ▶ Cumulative Time—The time spent inside this object and in other objects that the selected object invokes.
- ▶ Calls—The number of times the selected object was invoked by any other object.

Actions

Click the *Filter* icon  (second in the toolbar) to open the Filters window. There you can add, edit, or remove filters applied to the displayed results. For more information, refer to filtering (Figure 9-22 on page 292).

Click the *Columns* icon  (third) to open the Select Columns page. There you can select which columns are displayed in the table. These settings are saved with the current workspace, and are applied to all response time breakdown tables in the workspace.

Click the *Percentage* icon  (forth) to toggle the display between percentages and absolute values. In the percentage format, the table shows percentages instead of absolute values. The percentage figure represents the percentage of the total of all values for that column.

Click the *Source* icon  (fifth) to jump to the source code (if available) in your workspace. You must first select a method before clicking the source button.

Click the *Export* icon  (last) to open the New Report window. There you can export the response time breakdown table to CSV, HTML, or XML formats.

Use the navigation information in the upper left corner to navigate back to previous views. The navigation is presented in the form of a breadcrumb trail, making it easy to drill-down and drill-up from various reports.

In the past there has been some confusion around the terminology and values computed for Delivery Time and Response Time. If you are familiar with Performance Testing, you can see that Response Time here does not equate to a Performance Tester's definition of the term. The mathematical definitions are:

Delivery Time is Last_Received_Timestamp - First_Received_Timestamp

Response Time is First_Received_Timestamp - First_Sent_Timestamp

When viewing the transaction decomposition for a Graphics Interchange Format (GIF) file, you normally see a very small Delivery Time, often zero, because the entire data stream for the file arrives in a single packet. However, in a primary request of a page, if the server is taking a long time to respond, you typically see that the server responds with the header relatively quickly as compared to the remainder of the response, in which case the response might be broken down across several packets.

Note: You will always see a delivery time and a response time. You might also see transactions marked with the words DNS Lookup and Connect for requests that are of this nature.

Interactive Graphical Analysis

The interactive graphical analysis method involves a graphical drill-down process for decomposing a transaction using the existing Performance Tester report. If you choose to proceed toward response time breakdown analysis by selecting *Display Page Element Responses* (Figure 9-25 on page 295), you are presented with a bar-chart showing all the page element responses for the selected page (Figure 9-29).

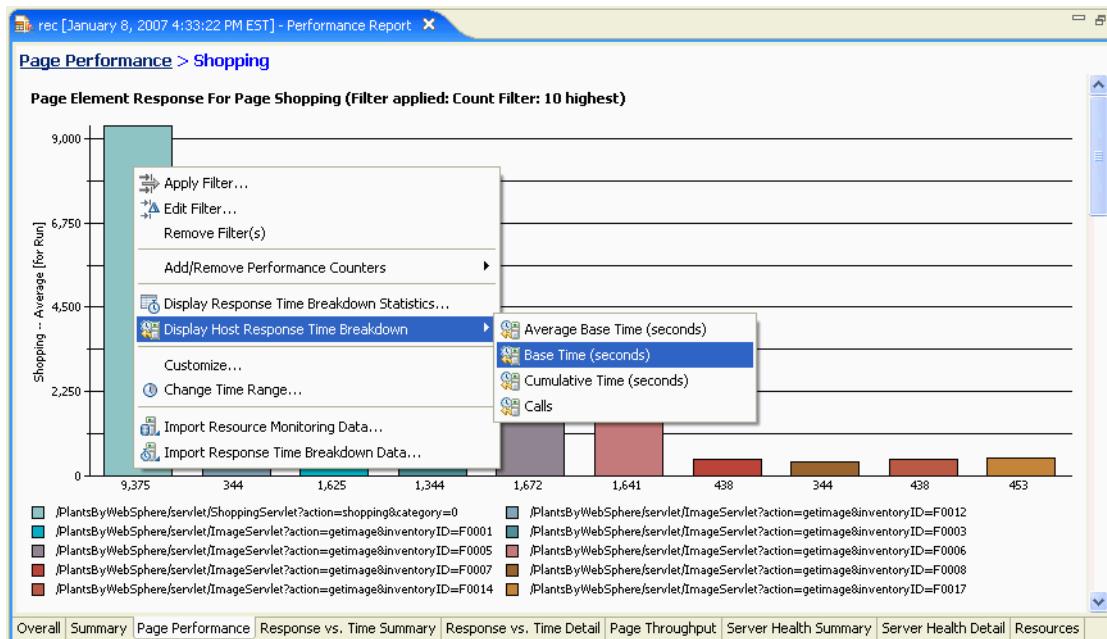


Figure 9-29 Graphical drill-down reports for Response Time Breakdown

From this page element drill down report, you can choose to further drill down into individual host responses, application and application component responses, and package, class, or method invocation responses for a particular element. The right-click context menu on the bar-graph displays which drill-down action is available from the current report. For example, if you are viewing the application response time breakdown for a particular transaction, the context menu will display a response time breakdown item for viewing the application components of the selected application (Figure 9-30).

At each level in the drill-down process, the breadcrumb path becomes more detailed. This behavior provides an easy way to jump between reports during the analysis process. You can also choose to jump directly to the Response Time Breakdown Statistics view from any drill-down graph.

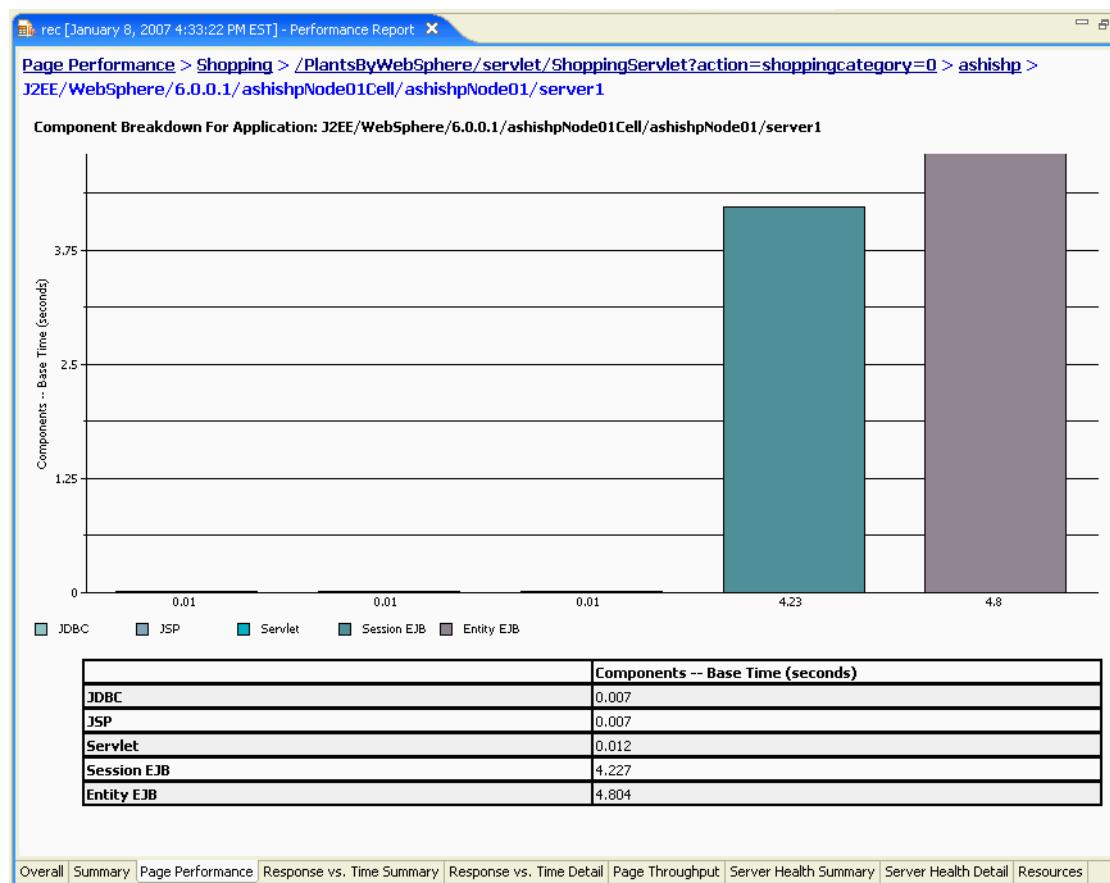


Figure 9-30 Component level drill-down report

9.3.5 Profiling for real-time observation analysis

The UML Interaction view presents a sequence of causal dependent events, where events are defined as method entries and exits, as well as outbound calls and return calls (Figure 9-31).

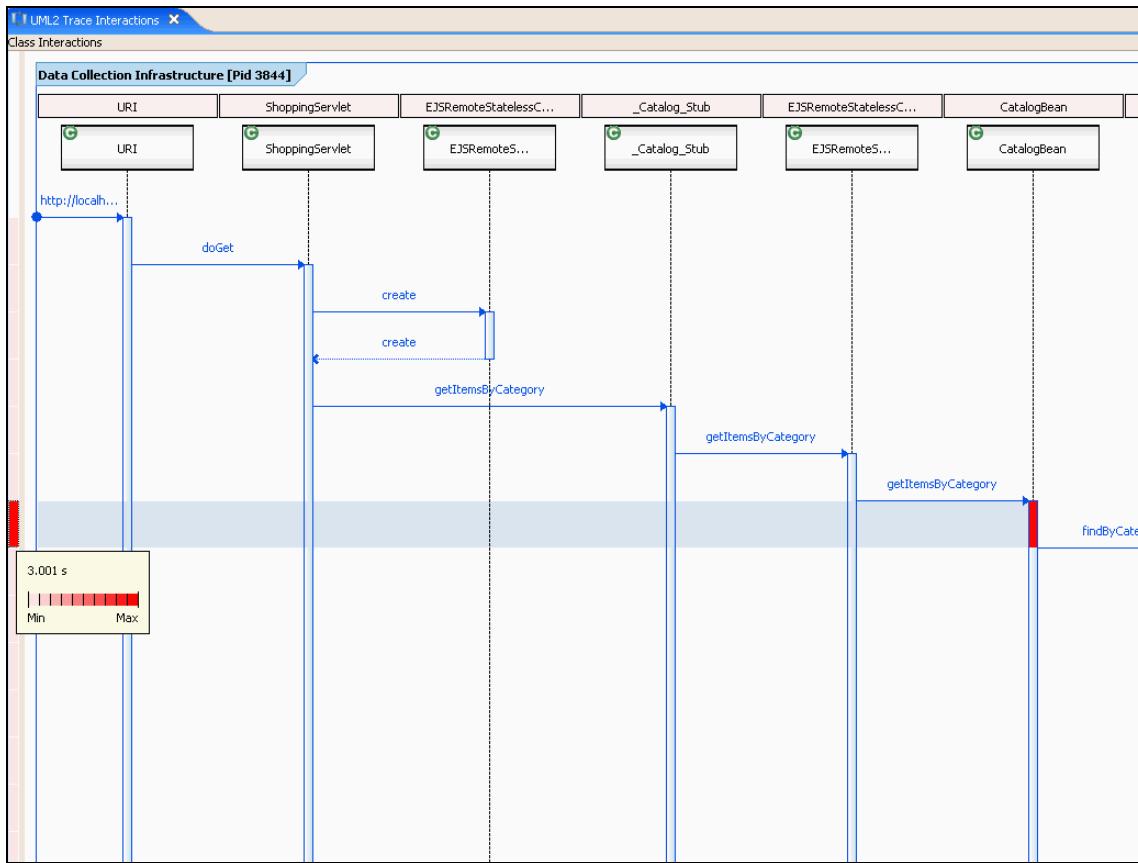


Figure 9-31 UML Sequence Diagram view

Specifically, it presents interactions between class instances. Those interactions have the form of method calls and call returns. The implementation of the Trace Interaction tool does extend that definition to one that generalizes actors of interactions as well as their means. In other words, the views provided by the tool are able to present not only the interactions of classes and class instances, but also those among threads, processes, and hosts. This extended use of the execution flow notation is motivated by the need to provide a hierarchy of data representation, which is necessary for large-scale, distributed traces.

A view linking service is available for assisting in correlating application trace events, as shown in the UML Sequence Diagram view, and Statistical and Log views (Figure 9-32). Correlation is computed using timestamps of two events. If an exact match is not found during correlation, the next event with the closest timestamp to the event being correlated is selected, within acceptable range.

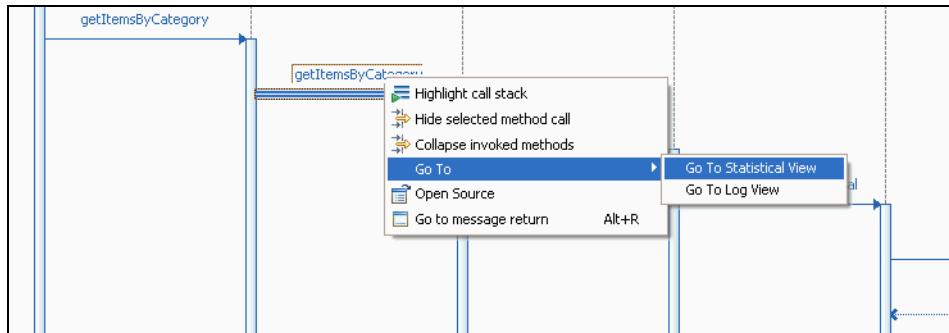


Figure 9-32 View linking service between UML sequence diagram and statistical or log

The Execution Statistics view displays statistics about the application execution time (Figure 9-33). It provides data such as the number of methods called, and the amount of time taken to execute every method. Execution statistics are available at the package, class, method, and instance level:

- ▶ Base Time—For any invocation, the base time is the time taken to execute the invocation, excluding the time spent in other methods that were called during the invocation.
- ▶ Average Base Time—The base time divided by the number of calls.
- ▶ Cumulative Time—For any invocation, the cumulative time is the time taken to execute all methods called from an invocation. If an invocation has no additional method calls, then the cumulative time will be equal to the base time.
- ▶ Calls—The number of calls made by a selected method.

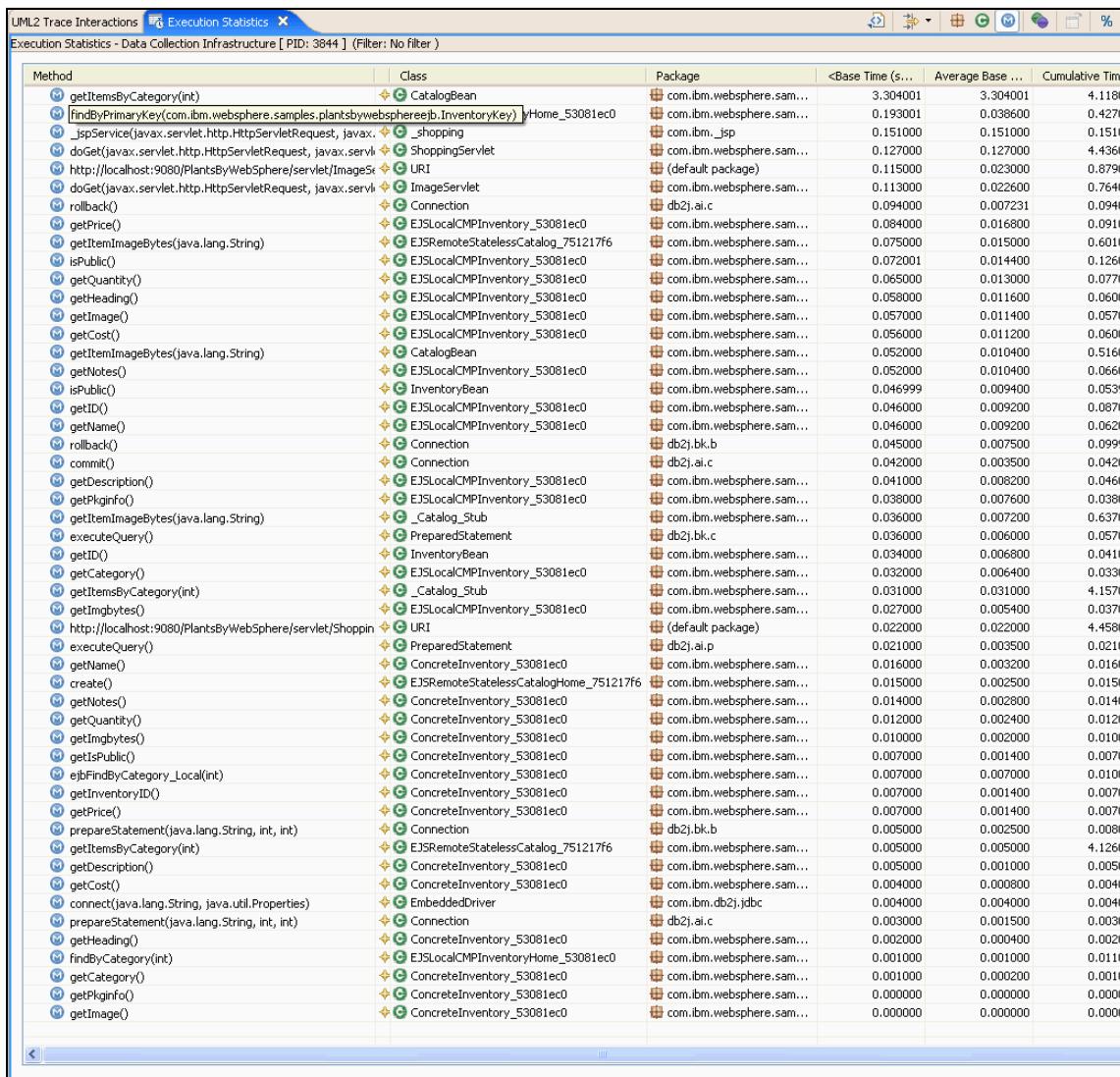
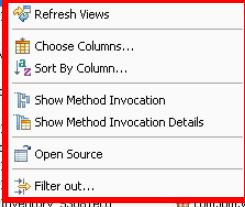


Figure 9-33 Execution Statistics view

Tip: Sort the entire list of methods by the Base Time metric. This action forces the method that consumes the most processing time to appear at the top of the list. From here, select the item and use the right-click menu to explore various analysis options.

Select an item from the Execution Statistics view and display the right-click context menu (Figure 9-34).



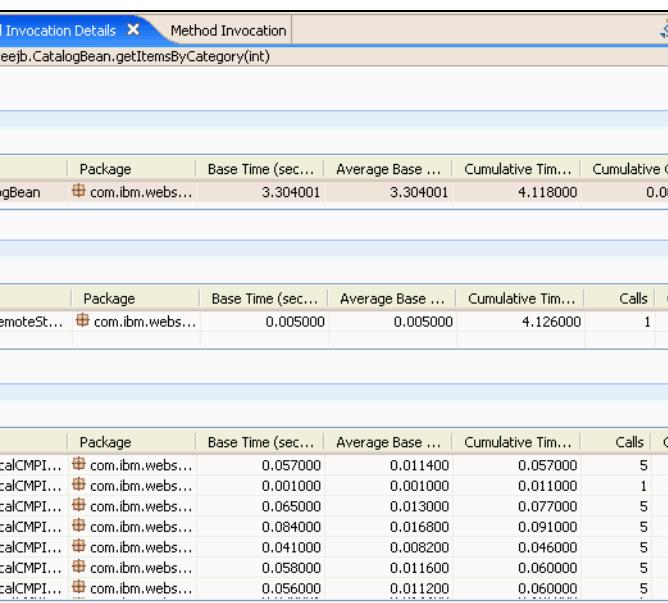
The screenshot shows the Execution Statistics view in a tool interface. A context menu is open over a row in the table, specifically for the method 'getItemsByCategory(int)'. The menu items are: Refresh Views, Choose Columns..., Sort By Column..., Show Method Invocation, Show Method Invocation Details, Open Source, and Filter out... . The 'Show Method Invocation Details' option is highlighted with a red box.

| Method | Class | Package | <Base Time (sec...) | Average Base ... | Cumulative Tim... |
|---|-----------------|------------------|---------------------|------------------|-------------------|
| getItemsByCategory(int) | CatalogBean | websphere.sam... | 3.304001 | 3.304001 | 4.118000 |
| findByPrimaryKey(com.ibm.websphere.samples.plantsbywe... | EJSLocalCMP... | websphere.sam... | 0.193001 | 0.036000 | 0.427000 |
| _jspservice(javax.servlet.http.HttpServletRequest, javax... | _shopping | sp | 0.151000 | 0.151000 | 0.151000 |
| doGet(javax.servlet.http.HttpServletRequest, javax.servl... | ShoppingServ... | websphere.sam... | 0.127000 | 0.127000 | 0.436000 |
| http://localhost:9080/PlantsByWebSphere/service/ImageSt... | URI | websphere.sam... | 0.115000 | 0.023000 | 0.879000 |
| doGet(javax.servlet.http.HttpServletRequest, javax.servl... | ImageServlet | websphere.sam... | 0.113000 | 0.022600 | 0.764000 |
| rollback() | Connection | websphere.sam... | 0.094000 | 0.007231 | 0.094000 |
| getPrice() | EJSLocalCMP... | websphere.sam... | 0.084000 | 0.016800 | 0.091000 |
| getImageBytes(java.lang.String) | EJSRemoteSt... | websphere.sam... | 0.075000 | 0.015000 | 0.601000 |
| isPublic() | EJSLocalCMP... | websphere.sam... | 0.072001 | 0.014400 | 0.126000 |
| getQuantity() | EJSLocalCMP... | websphere.sam... | 0.065000 | 0.013000 | 0.077000 |
| getHeading() | EJSLocalCMP... | websphere.sam... | 0.058000 | 0.011600 | 0.060000 |

Figure 9-34 Execution Statistics view right-click context menu

In this menu there are various actions that can be executed, notably:

- ▶ Open Source—Allows the user to automatically open the source code associated to the selected package, class, and method combination if the source is imported into the current workspace.
- ▶ Method Invocation Details (Figure 9-35)—Provides statistical data on a selected method including information about the invoker method and the methods invoked by the selected method. This view can be opened on a selected method in any of the profiling views, such as the Method Invocation Details, Execution Flow, or Execution Statistics view.



The screenshot shows the Method Invocation Details view. At the top, the tabs are UML2 Trace Interactions, Execution Statistics, Execution Flow, Method Invocation Details (which is active), and Method Invocation. Below the tabs, the title is 'Method Invocation Details: com.ibm.websphere.samples.plantsbywebspherewe...ejb.CatalogBean.getItemsByCategory(int)'. The view is divided into three main sections: Selected method, Selected method is invoked by:, and Selected method invokes:.

Selected method:

| >Calls | Method | Class | Package | Base Time (sec...) | Average Base ... | Cumulative Tim... | Cumulative C... |
|--------|-------------------------|-------------|-----------------|--------------------|------------------|-------------------|-----------------|
| 1 | getItemsByCategory(int) | CatalogBean | com.ibm.webs... | 3.304001 | 3.304001 | 4.118000 | 0.00 |

Selected method is invoked by:

| >Invoked by | Method | Class | Package | Base Time (sec...) | Average Base ... | Cumulative Tim... | Calls |
|-------------|-------------------------|----------------|-----------------|--------------------|------------------|-------------------|-------|
| 2 | getItemsByCategory(int) | EJSRemoteSt... | com.ibm.webs... | 0.005000 | 0.005000 | 4.126000 | 1 |

Selected method invokes:

| >Invokes | Method | Class | Package | Base Time (sec...) | Average Base ... | Cumulative Tim... | Calls |
|----------|---------------------|----------------|-----------------|--------------------|------------------|-------------------|-------|
| 10 | getImage() | EJSLocalCMP... | com.ibm.webs... | 0.057000 | 0.011400 | 0.057000 | 5 |
| 2 | findByCategory(int) | EJSLocalCMP... | com.ibm.webs... | 0.001000 | 0.001000 | 0.011000 | 1 |
| 10 | getQuantity() | EJSLocalCMP... | com.ibm.webs... | 0.065000 | 0.013000 | 0.077000 | 5 |
| 10 | getPrice() | EJSLocalCMP... | com.ibm.webs... | 0.084000 | 0.016800 | 0.091000 | 5 |
| 10 | getDescription() | EJSLocalCMP... | com.ibm.webs... | 0.041000 | 0.008200 | 0.046000 | 5 |
| 10 | getHeading() | EJSLocalCMP... | com.ibm.webs... | 0.058000 | 0.011600 | 0.060000 | 5 |
| 10 | getCost() | EJSLocalCMP... | com.ibm.webs... | 0.056000 | 0.011200 | 0.060000 | 5 |

Figure 9-35 Method Invocation Details view

The Method Invocation Details view describes:

- ▶ Selected method—Shows details including the number of times the selected method is called, the class and package information, and the time taken.
- ▶ Selected method invoked by—Shows details of each method that calls the selected method, including the number of calls to the selected method, and the number of times the selected method is invoked by the caller.
- ▶ Selected method invokes—Shows details of each method invoked by the selected method, including the method name and the number of times the method is invoked, package and class information of the invoked method, and the time spent ion the invoked method.

9.4 Collecting application monitoring data from IBM Tivoli family of products

An area of the ITLM cycle that has a large void between is the space after an application has been deployed into production and the development team that creates the applications. Consider the following scenario:

- ▶ An IT operator becomes aware of a performance problem with the system, either through a manual observation with a monitoring tool, or a customer complaint. Through analysis of the operators production tool reports, the operator determines that the performance problem only occurs when users are causing a heavy load on the system.
- ▶ The IT operator sends information to the development team.
- ▶ The development team attempts to fix the problem.

A primary issue with the information that is sent from an IT operator to a developer is that it is not always understood by a developer to its full extent. This problem can be solved by creating tools and reports that an IT operator and a developer understand within their own domains, while using the same data source. Therefore, the data available to both parities is consistent, but presented in a manner that can be understood by the specific user.

IBM Tivoli Composite Application Manager (ITCAM) for Response Time Tracking and ITCAM for WebSphere are two products that are used in the operations space. These two products collect extensive information when monitoring applications in production. To take this information and provide meaning to it for a developer, a transformation can be placed on the data using Performance Tester. Therefore, when a problem is identified by an IT operator using ITCAM, a developer can use Rational development and test tools to import the data from ITCAM. This action of importing data transforms the data and presents it to the user in their terminology.

9.4.1 Architecture

Acquiring data from the ITCAM Management Server is achieved through a well-defined Web Service Definition Language (WSDL) called *DataQuery* (Figure 9-36).

| DataQueryRemote | | |
|--|--|--------------|
| getManagementPolicies | | |
|  input | in0 | long |
| | in1 | long |
|  output | getManagementPoliciesReturn | Vector |
| | | |
| getPolicyStatusOnHostForPolicies | | |
|  input | in0 | Vector |
| | getPolicyStatusOnHostForPoliciesReturn | Vector |
| getInstanceRootDetails | | |
|  input | in0 | int |
| | in1 | long |
|  input | in2 | long |
| | getInstanceRootDetailsReturn | Vector |
| getInstanceTopology | | |
|  input | in0 | int |
| | in1 | InstanceRoot |
|  output | getInstanceTopologyReturn | Vector |
| | | |
| getAggregateTopology | | |
|  input | in0 | int |
| | in1 | long |
|  input | in2 | long |
| | getAggregateTopologyReturn | Vector |

Figure 9-36 DataQuery Web service API

This Web service defines the following functions:

- ▶ **getManagementPolicies**—Returns a list of all the configured and active Management policies, which describe the constraints for transactions executed on a given application being monitored.
- ▶ **getPolicyStatusOnHostForPolicies**—Returns the current status of a policy. Various status indicators, such as, OK, Critical, Warning, Fatal help to indicate the severity of a transaction's recent behavior, given a policies constraints.
- ▶ **getInstanceRootDetails**—Returns the transaction details for each instance of a particular transaction.
- ▶ **getInstanceTopology**—Returns the entire transaction topology of a given root transaction.
- ▶ **getAggregateTopology**—Returns the aggregate of a transaction's topology over a specific period of time.

The Web service returns wrapper objects that contain the same ARM objects that were discovered at runtime of the transaction. Similar to the real-time data collection scenario, these ARM objects are organized into a transactional hierarchy, from the root transaction to all of its sub-transactions. This hierarchy is then converted into a set of events that are modeled after the TPTP trace model format. This transformation is executed at the client-side of the environment (Figure 9-37). When the transformation is complete, the same user interfaces used for analysis during real-time data collection are available here.

This Web service interface also handles all authentication and authorization through standard Web service authentication and authorization means.

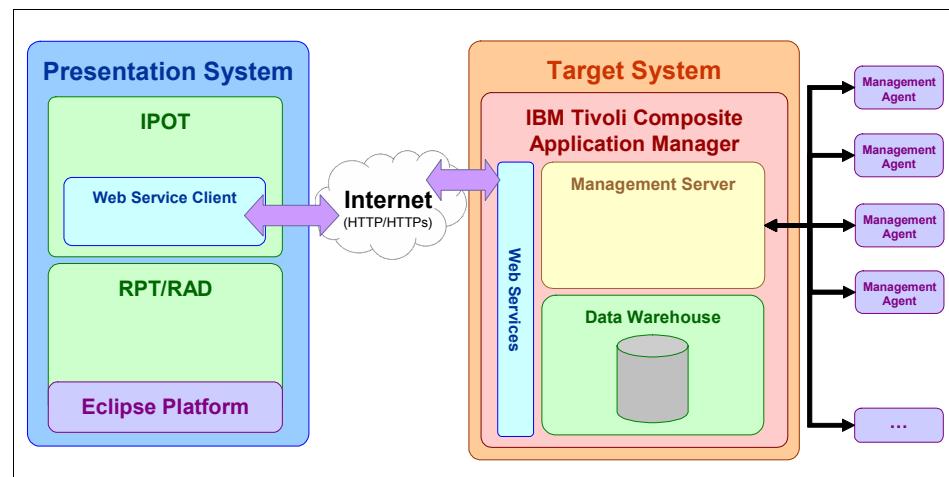


Figure 9-37 System architecture using the DataQuery Web service

9.4.2 Importing application monitoring data

ITCAM for Response Time Tracking and ITCAM for WebSphere are managed solutions that store all observed application trace data in a data warehouse.

To import response time breakdown data, one of the IBM Tivoli Monitoring products must be installed on the application server that collects the application data.

The following ITCAM configuration options can affect what data is available to be retrieved and imported:

- ▶ ITCAM has two modes of data collection: aggregated and instance.
 - *Aggregated* statistical data consists of high-level statistics about classes and methods, such as average execution time. Aggregated statistics might not reveal intermittent performance problems.

- *Instance-level* statistics consist of data on a per-instance basis in addition to aggregated statistics. That is, you can see exactly what calls were made, by which process, and at what time. The data stored in the database depends on how ITCAM is configured prior to data collection.
- ▶ ITCAM can also be configured either to record instance-level data all the time, or to start collecting only when a threshold has been violated.
- ▶ In ITCAM, trace levels are configurable (off, low, medium, high, custom), and the configuration setting affects how much data is collected by the IBM Tivoli Monitoring server and stored in the database. The performance analysis tools will import all of the data collected with respect to this trace configuration setting. If the ITCAM setting is too low, you might collect less data than you need and expect.

In the workbench client, you can import the data using two methods:

- ▶ An Import wizard that is accessible by selecting *File* → *Import* (Figure 9-38).
- ▶ An Import wizard that is accessible by right-clicking on a performance report in the Performance Test Runs view (Figure 9-39) or on the resources report, which is the last tab in the view.

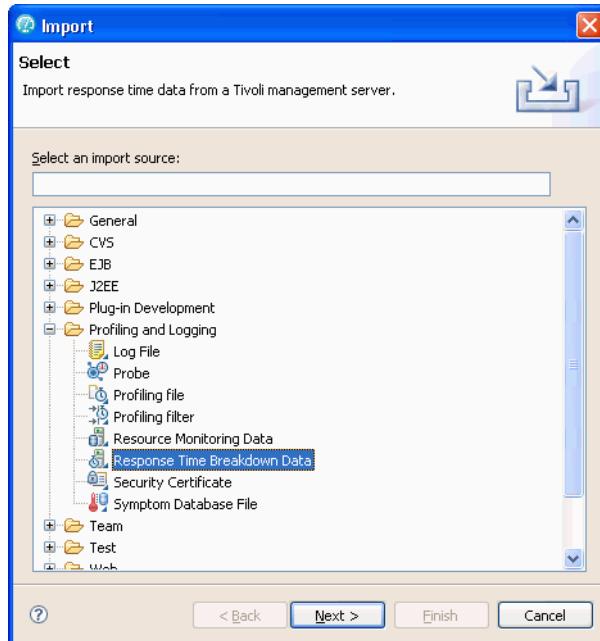


Figure 9-38 Performance Tester Import wizard

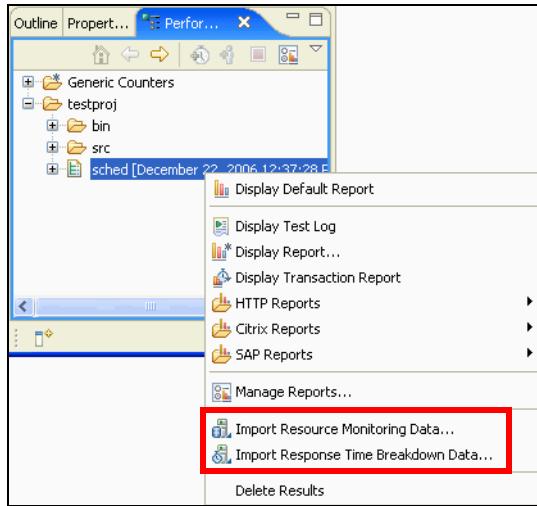


Figure 9-39 Performance result context menu

Import wizard

On the first page of the Import wizard, in the Host field, you must specify the host of the ITCAM Management server (Figure 9-40).

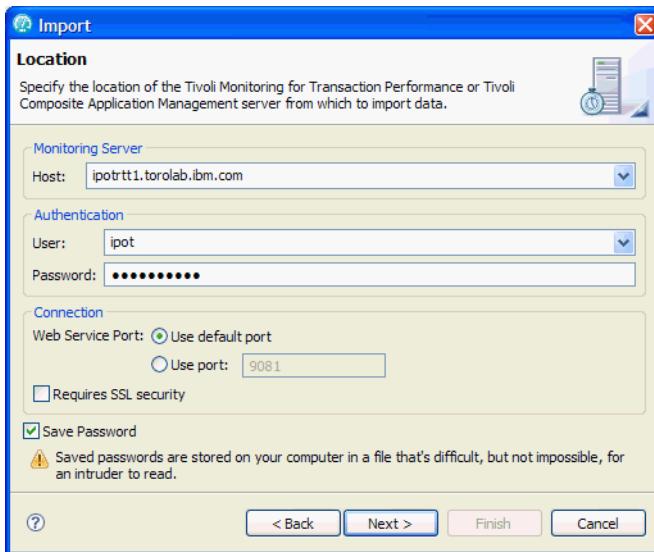


Figure 9-40 Response Time Breakdown Import wizard

Follow these steps:

- ▶ If you use a user ID and password to log on, type them into the User and Password fields.
- ▶ If the server is configured to use Secure Sockets Layer (SSL), select *Requires SSL security*.
- ▶ The ports are used to transmit the data. If you select *Use default port* for the Web Service Port, it will use the default ports, 9081 without security or 9446 with SSL. If the administrator has configured the ports differently, select *Use port* and specify the port that the administrator has configured.

Next, specify the time period for which to import data and the type of data (Figure 9-41).

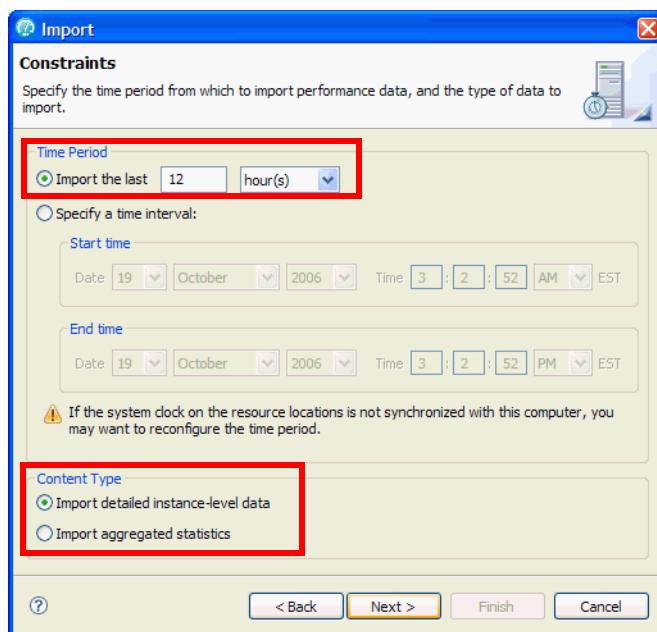


Figure 9-41 Response Time Breakdown Import constraints

- ▶ You might have to adjust the time interval to compensate for clock discrepancies between the workbench and the management server.
- ▶ Specify the type of data that you want to import. For detailed analysis, you would typically want to select *Import detailed instance-level data*, although it might involve a large quantity of data. If you select *Import aggregated statistics*, the data imported will be statistical summaries, not the kind of data that can be analyzed in sequence diagrams or trace tools.

Notes:

- ▶ When specifying the time in a specific number of units, for consistency, *month* is defined as 30 days and *year* is defined as 365 days. *Days* refers to 24-hour periods, not calendar days. The units of time selected are subtracted from the point in time when you click *Finish* to import the data to give the start time. For example, if you select 2 months, the time period will be the 60 days (24-hour time periods) immediately prior to the point in time that you click *Finish*.
- ▶ It is important to test the smallest possible portion of the application to focus only on the part that exhibits the problem. This will simplify the analysis later. If you are unsure of what the trigger is, you might have to collect more data and use more thorough filters to view the data.
- ▶ IBM Tivoli Composite Application Manager for Response Time Tracking must be configured to record instance-level data to import instance-level data. If IBM Tivoli Composite Application Manager for Response Time Tracking is configured to collect only aggregated data, regardless of what you select here, only aggregated statistical data will be available in the IBM Tivoli Composite Application Manager for Response Time Tracking database. IBM Tivoli Composite Application Manager for WebSphere does not collect aggregated statistics. If you specify Import aggregated statistics with IBM Tivoli Composite Application Manager for WebSphere, the import will fail.

Select the policies or traps from which you want to import data (Figure 9-42). Typically, you will import from policies or traps that have a severe problem status (such as critical or failure), because they will point you to the code problems. Selecting only the most serious policies or traps helps limit the quantity of data imported. Click the *Status* column heading to sort the policies by severity.

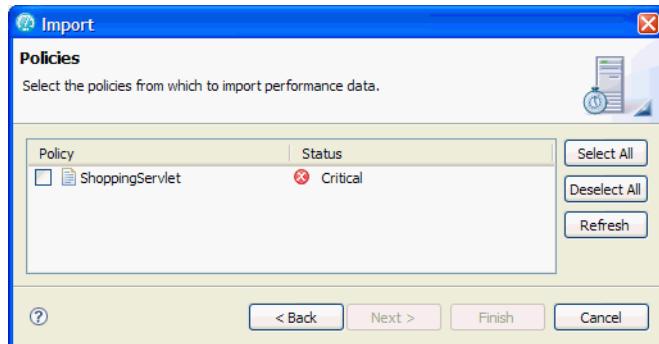


Figure 9-42 Response Time Breakdown Import policy selection'

Policies are used by IBM Tivoli Monitoring for Transaction Performance and IBM Tivoli Composite Application Manager for Response Time Tracking. Traps are used by IBM Tivoli Composite Application Manager for WebSphere.

The policies, or traps, that you selected are associated with a group of hosts (computers) in the application system. Select the hosts that you want to examine (usually the hosts with the more critical status). Specifying a subset of hosts will again reduce the quantity of data imported, helping you focus on where the problems really are (Figure 9-43).

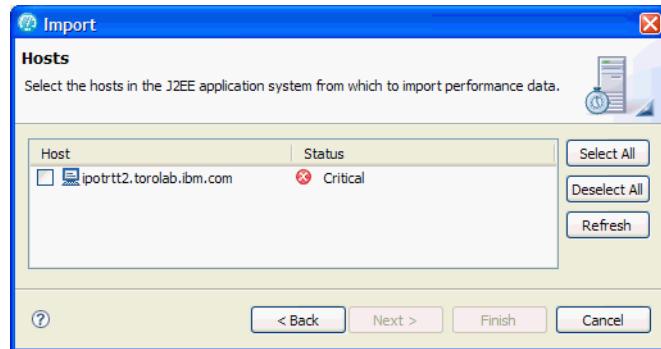


Figure 9-43 Response Time Breakdown Import host selection

When importing data from ITCAM for WebSphere, the WebSphere Application Server node name is listed, rather than the actual host name. This is set by the server administrator and does not necessarily have to be related to the actual host name, though that is the default choice.

If you chose to import instance-level data, you can select the transactions for which you are interested in seeing data (Figure 9-44).

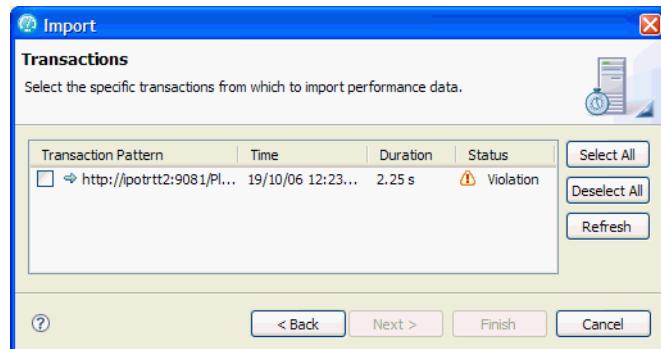


Figure 9-44 Response Time Breakdown Import transaction selection

The Time column indicates when the transaction started, and the Duration column indicates how long the transaction took. You can sort the data by clicking on any of the column headings.

The transaction pattern listed is not the actual URL of the transaction, but rather a regular expression pattern match for the URL that it matched in ITCAM.

Note: You can configure the IBM Tivoli Composite Application Manager for Response Time Tracking listening policy to collect hourly averages by unique transaction, or by monitor. You must set the listening policy to *Hourly Average by Unique Transaction* to see full transaction names.

After you have collected the response time breakdown data, you can begin analyzing it and diagnosing the problem. You can view the data using several views (Figure 9-45), including statistics views and sequence diagrams of class and object interactions (Figure 9-46), to help find the cause of the performance problem.

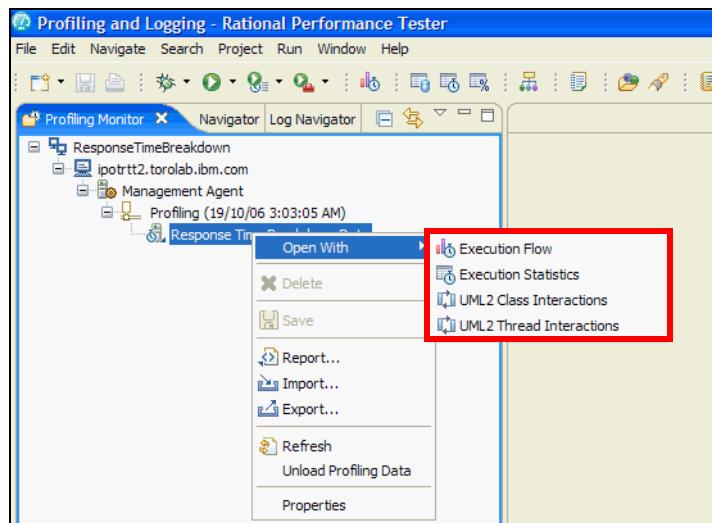


Figure 9-45 Associated views with the management agent

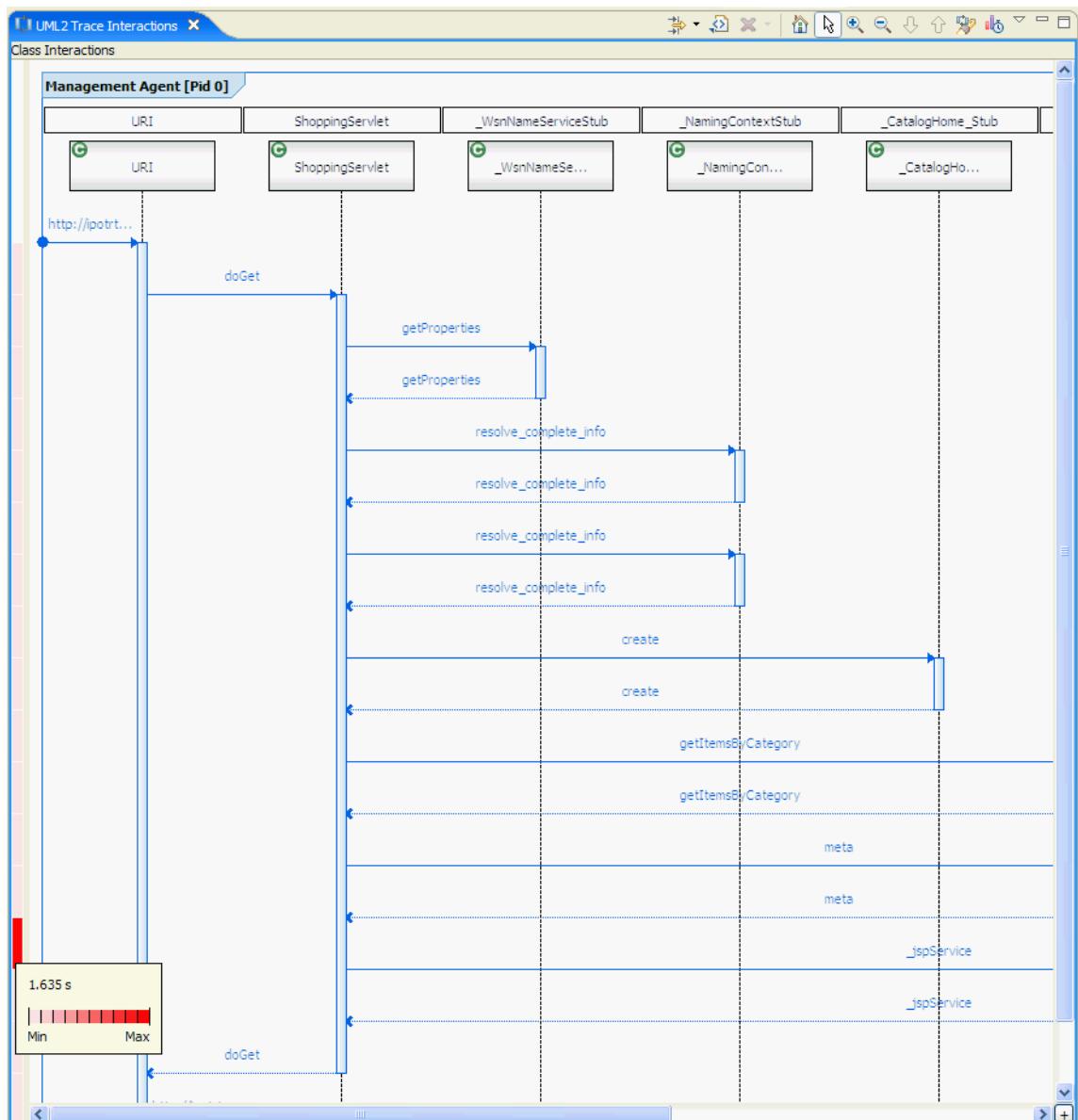


Figure 9-46 UML Sequence Diagram view for data imported from ITCAM

Note: When importing response time breakdown data from ITCAM for WebSphere, there are two authentication layers involved. The first one is WebSphere authentication, which will reject an invalid user/password on the system and display an authentication dialog. The other is ITCAM for WebSphere authentication, which will simply return no data available to import if authentication fails.

The only case where WebSphere authentication will pass and ITCAM for WebSphere authentication will fail is when the user types a valid user name on the underlying operating system (for example, root), but that user is not registered in ITCAM for WebSphere. In this case, the user must be aware that the server will not display an error when authentication fails, but the user will instead see no traps available from which to import.

When importing data from an ITCAM for WebSphere trap, ensure that the clocks of the monitoring server and the workbench are synchronized. In the import wizard, the option to import the last *n* units of time uses the current time on the local computer, but queries for traps which have activity in that time period on the monitoring server clock. So if the monitoring server clock is ahead by 10 minutes, you will have to either wait 10 minutes before the import wizard will find this transaction available on the server, or query 10 minutes into the future.

Methods are timed in ITCAM for WebSphere using millisecond granularity. Because of this, when importing from the ITCAM for WebSphere monitoring server, the resulting data might show response times of 0 for certain methods with extremely fast execution.

Class names that are stored in method traces by IITCAM for WebSphere might be truncated when you import data from the ITCAM for WebSphere monitoring server. To avoid truncation, make the following changes to aa.properties, aa1.properties, and aa2.properties under MS_HOME/etc:

```
TEXTSTRING_LENGTH=255
```

Where MS_HOME is the installation home directory of the ITCAM for WebSphere monitoring server.

9.5 More information

For more information, refer to these additional resources:

- ▶ The Redbooks publication, *End-to-End e-business Transaction Management Made Easy*, SG24-6080
- ▶ Eclipse Test & Performance Tools Platform documentation:
<http://www.eclipse.org/tptp>

Applying RPT to enterprise application testing

The theme for this part of the book is to explain:

- ▶ **How can RPT can be applied or extended to test significant enterprise environments?**

We introduce various tool usage techniques determined by characteristics of the particular environment under consideration.



Scalability and test agents

This chapter provides hardware sizing guidelines for IBM Rational Performance Tester, and provides insight into what factors affect memory footprint and CPU performance. Although the measurements were made using RPT 6.1.2, the values for RPT 7.0.0.x releases should fall within a tolerance of +/- 10-15%, allowing the guidelines to be applied to RPT 7.0.0.x releases as well as 6.1.2.

The scope of this chapter is the Windows platform, covering both the workbench console and driver (test agent) machine considerations, but focusing mostly on the driver because that is the primary variable in hardware sizing. We only cover the HTTP protocol; Siebel, SAP, and Citrix protocols are not covered.

Rational Performance Tester 6.1.2 Workbench Memory Optimization

This chapter references IBM technote 1221972: *Rational Performance Tester 6.1.2 Workbench Memory Optimization*. It is applicable to RPT 7.0, except for instructions on how to set the workbench JVM's maximum heap size; however, be aware of the terminology change introduced in 7.0 (execution history = test log). To locate this document from within RPT:

- ▶ Select *Help* → *Search* and search for *Workbench Memory Optimization*.
- ▶ Select the link under IBM developerWorks named Web Search.
- ▶ Open the RPT 6.1.2 Workbench Memory Optimization tech note.

You can also find the technote on the Web at:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21221972>

10.1 Introduction

In this section we provide an overview of the high-level process architecture of RPT, the workbench and driver architecture, and Java memory management considerations.

This section is introductory in nature and provides useful context for assimilating and interpreting the measurement results and sizing guidelines. Users interested in skipping the details and getting to the main points should at least browse 10.2, “Driver measurements” and then read 10.3, “RPT sizing guidelines” in its entirety. The last three sections are supplemental and can be treated as an appendix for those wishing to apply or deepen their understanding of the preceding material.

10.1.1 Abbreviations

The following abbreviations are used in this chapter:

- ▶ **EMF**—Eclipse Modeling Framework.
- ▶ **Driver**—The collection of components that execute a test. Equivalent to **test agent**, **agent**, or **injector**. Driver often refers to the machine on which the test is executed, especially if the workbench is located on a different machine.
There can be multiple drivers associated with a test run that is initiated from a workbench.
- ▶ **Incremental virtual user memory footprint**—The memory cost on the driver of adding one additional virtual user to the workload mix.
- ▶ **Incremental virtual user CPU utilization**—The CPU utilization cost, measured as a percentage, on the driver of adding one additional virtual user to the workload mix.
- ▶ **JVM**—Java Virtual Machine, the process which executes Java byte code.
- ▶ **KB**—Kilobytes, equivalent to 1024 bytes.
- ▶ **MB**—Megabytes, equivalent to 1,048,576 (1024 * 1024) bytes.
- ▶ **mB**—Million bytes, equivalent to 1,000,000 bytes.
- ▶ **RPT**—IBM Rational Performance Tester.
- ▶ **TPTP**—Test & Performance Tool Platform (an open source Eclipse project, refer to <http://www.eclipse.org/tptp>).
- ▶ **Workbench**—The Eclipse workbench process, where the RPT user interface is hosted. Workbench sometimes refers to the machine on which this process runs, to distinguish it from the driver machine, if they are different. The workbench is also referred as **controller** or **console**.

10.1.2 RPT process architecture

Figure 10-1 shows the RPT process architecture, depicting machine, process, and JVM boundaries.

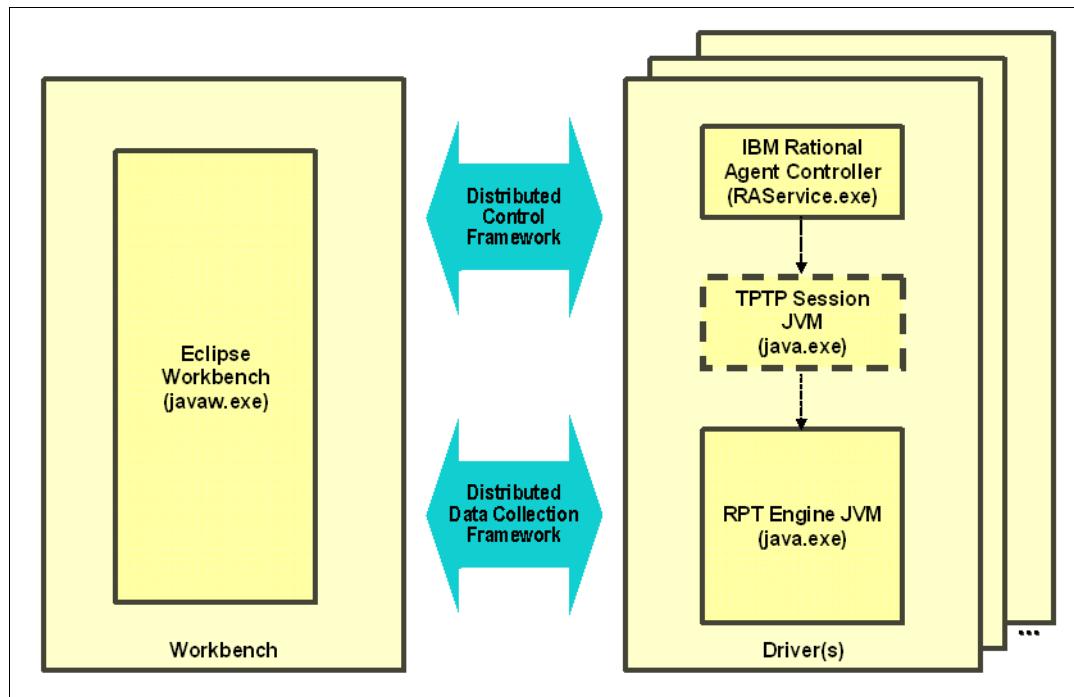


Figure 10-1 RPT process architecture (machine, process, and JVM boundaries)

There can be an arbitrary number of drivers, one per machine. One driver can be located on the same machine as the workbench. The Eclipse workbench process (`javaw.exe`) on the workbench machine hosts the user interface for RPT. Virtual user execution is performed in the RPT engine JVM process (`java.exe`) on the driver(s), which is the primary focus of this document. In TPTP terminology, this engine process is also typically referred to as the test runner. The TPTP session JVM process is a temporal process that exits upon successful creation of the engine, so it does not factor into the RPT driver memory footprint.

Multiple workbenches can concurrently share the same driver(s), although this is not recommended for production load testing runs, which should normally be performed on dedicated hardware. There also can be multiple RPT engine JVMs per machine when driven from a single workbench, but this is not normal, and requires special configuration knowledge.

10.1.3 RPT workbench architecture

The functions performed by RPT on the workbench are determined by the Eclipse TPTP architecture, which RPT is based on, and are shown in Figure 10-2.

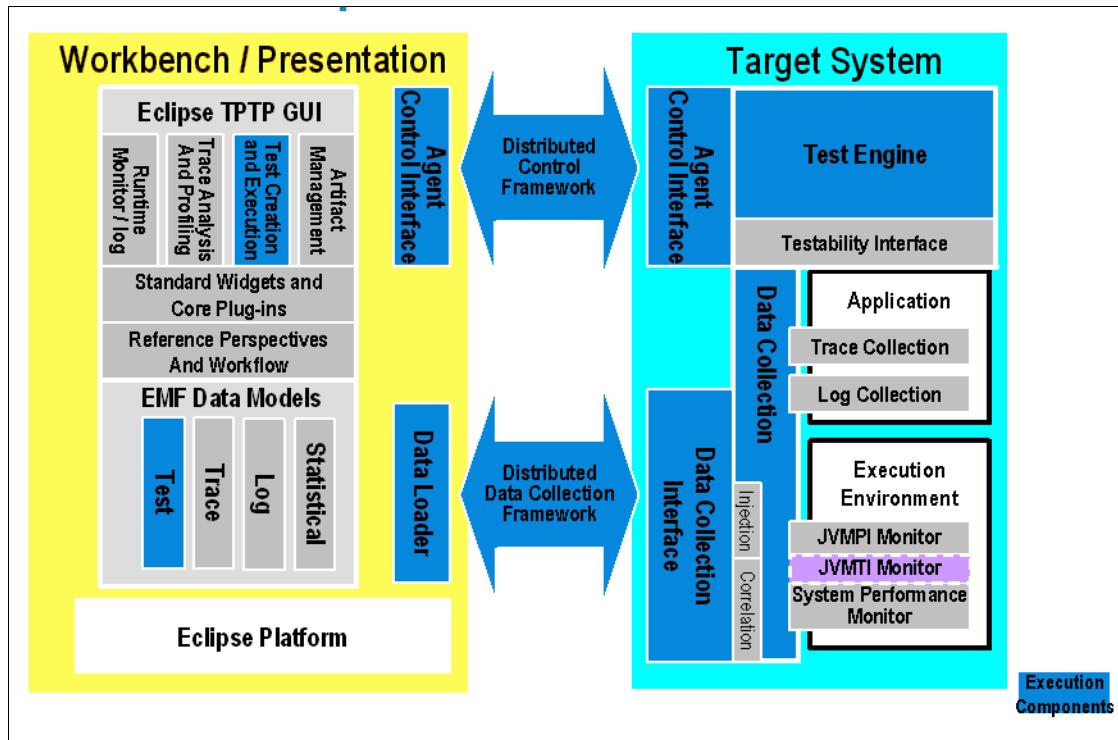


Figure 10-2 Eclipse TPTP architecture

As shown in Figure 10-2, the workbench not only provides the GUI presentation face of RPT for all views and editors, test creation and execution, and artifact management, but it is also where the TPTP EMF data models are loaded. EMF is an in-memory modeling technology and, unless special approaches are utilized for a particular EMF model, all of the data stored in an EMF model must be loaded into memory in its entirety if any part of the model is to be read (for example, when a report is opened). In the case of RPT, the EMF data models are the primary consumer of workbench memory, and in particular the statistical model and the execution history in the test model.

10.1.4 RPT driver architecture

As shown in Figure 10-1 on page 323, the driver consists of three processes: a service process (RAService.exe) that is always present; a temporal session process (java.exe) created for each schedule/test run that is irrelevant for driver sizing because it immediately exits when the RPT engine JVM process is started; and the RPT engine JVM (java.exe) process.

The RAService.exe process memory footprint is best treated as constant overhead, independent of the size of the run, and present whether a run is active or not. The memory footprint of RAService.exe is a relatively constant amount, approximately 20 MB. It is not a material factor in sizing considerations, and we do not discuss it further.

RPT engine architecture

The RPT engine JVM (java.exe) is the single process per driver that generates the virtual user load against the server(s) under test, and whose memory footprint and CPU overhead grow as the number of users is increased or the activity (workload) is increased. The architecture supports an arbitrary number of protocols, which in and of themselves do not require separate processes or threads.

There are several notable aspects of the RPT engine's patent-pending architecture that are related to its memory and CPU scalability characteristics that are called out here:

- ▶ Perhaps the most obvious is that the engine is implemented in Java, and runs in the JVM, resulting in a set of inherent characteristics and considerations, as discussed in 10.1.5, "Java memory management considerations" on page 326.
- ▶ The RPT engine is based on a hyper-threaded, object-oriented, queued design that does not require an operating system thread per user. Instead, it has a fixed (but tunable) small number of *worker* threads (30) that perform the actions present in the executing performance test(s), which in practice sufficiently covers a wide range of loads into the thousands of users without introducing significant queuing delays. There is also another fixed set (just under 30 in the current RPT 6.1.2 implementation) of service threads that perform various services on behalf of the worker threads.
- ▶ The RPT engine I/O subsystem performs efficient asynchronous I/O, based on the non-blocking capabilities of the Java NIO interface.
- ▶ The combination of the hyper-threaded, queued design and the asynchronous I/O provide a very highly scalable architecture that greatly reduces the operating system overhead in CPU and memory (housekeeping and per-thread stack/heap data) associated with thousands of threads in a

conventional load testing implementation that employs a thread for each user. When a virtual user is not performing an action, but is *thinking* or waiting for I/O from the server, the RPT engine architecture consumes no CPU overhead and almost zero memory overhead. Because idle time spent waiting often takes up the vast majority of time of a virtual user, RPT's memory footprint can be extremely low for workloads that have very high numbers of relatively inactive users.

10.1.5 Java memory management considerations

A Java implementation of the workbench and driver engine presents fundamentally different considerations for memory management and sizing than a native code implementation. The most significant aspects are introduced in this section, as applicable to the RPT driver engine. Although similar, if you are interested in aspects specific to the workbench, refer to the related document, *Rational Performance Tester 6.1.2 Workbench Memory Optimization*.

JVM heap limits

The Java Virtual Machine (JVM) provides heap space for all memory allocated by the Java code which executes within the JVM. Essentially a Java implementation means available memory is limited to the JVM maximum heap size, *regardless of the amount of memory on the system*. In other words, the user must manage the maximum amount of memory to be allocated to each Java virtual machine and—unlike native code—cannot rely on the operating system's physical and virtual memory management to manage memory automatically.

A 32-bit JVM naturally restricts the maximum heap to somewhat less than 2 GB, but there can be additional restrictions, some of which might vary by operating system. In practice, the IBM 1.4.2 Sovereign JVM might not successfully initialize on some system configurations when the maximum heap is set larger than 1900 MB, so this becomes the default practical limit for the RPT driver engine. (As of RPT 7.0, the version of the IBM JVM has been updated to 1.5 and the practical maximum heap value is 1500 on Windows and 3000 on Red Hat Linux. Accordingly, RPT uses a default maximum of 1500 with RPT 7.0 when automatically setting the JVM max heap value.)

On the first playback on a driver, RPT by default sets the maximum heap size for the driver JVM equal to 256 MB and then—on all operating systems except Windows 2000—automatically adjusts the JVM heap size on subsequent runs to be 256 MB less than the remote driver machine's physical memory present during the previous run.

This value is persisted in the driver's location asset, under the attribute RPT_DEFAULT_MEMORY_SIZE. (It is reset every run, so it is fruitless to edit it!) When running users locally, unless you explicitly create a location for the local driver and reference it in the schedule's user group when assigning users, the automatic calibration is not available and the default 256 MB maximum heap size is used for the local driver every run, not just the first. You can override RPT's algorithms for setting driver heap size, as outlined in 10.4, "Setting driver maximum JVM heap size" on page 364.

If the maximum JVM heap size is exceeded, the java.exe engine process will be unable to process further actions (whether or not the JVM successfully exits). The workbench will detect the absence of a heartbeat from the engine process and will, in approximately 60 seconds, present an error dialog to the end user, and terminate the run. The same workbench *non-responsive driver* determination might also occur due to excessive garbage collection overhead as the driver JVM nears heap exhaustion or if it is producing a core file. In these cases, run termination might precede the actual java.exe termination. The bottom line is that exhaustion of the JVM heap will terminate the run. It can be avoided in subsequent runs by increasing the maximum heap allocation, reducing the number of users on that driver, or reducing the total workload size.

JVM memory heap allocation and garbage collection complexities

It is beyond the scope of this paper to present in any depth the subject of JVM heap allocation and garbage collection. Algorithms and options that control garbage collection behavior and decisions by the JVM as to when to grow the heap and request additional memory from the operating system versus when to perform garbage collection vary by JVM vendor, version, and invocation options. For example, the IBM 1.4.2 j9 JVM used by Eclipse in the IBM Software Delivery Platform (SDP) products (for example, RPT and Rational Application Developer) has different characteristics than the IBM 1.4.2 Sovereign JVM used by RPT's driver engine. Further discussion here is limited to what is applicable for the driver memory optimization, which uses the IBM 1.4.2 Sovereign JVM.

There is an occasional dramatic rise in JVM heap usage from a firmly established steady state level for no apparent reason, often constrained only by the max heap value, and frequently, but not always, followed by an eventual, gradual drop-off, although typically never down to the original steady state level. The process sometimes repeats itself. Although the exact cause of this behavior has not been determined, it is attributed in general terms to the JVM's penchant for occasionally allocating memory much more freely than actually needed.

Upon close study, this behavior is shown to be nothing other than an *artificial JVM phenomena* that is not reflective of any real steady state memory need of the virtual users; repeated subsequent runs with smaller and smaller constrained heaps show that the bulk of the exaggerated memory allocation disappears under the much more shallow maximum heap with no ill effect (such that an ideally constrained heap would eliminate them). In any case, this *voom* effect can give an inflated perception of driver memory consumption if you take a periodic size check and use that value indiscriminately in projecting memory needs for larger user loads. For this reason, the best practices for measuring and projecting incremental per-user memory usage outlined in this document always involve observing memory over time, and using steady-state values that occur prior to the occasional grossly inflated values as described previously.

Note that the Java heap is the predominant—but by no means the exclusive—contributor to the JVM process memory size. The JVM implementation itself requires memory for its own code, stack, and data (static and dynamic). For example, a 1024 MB max heap (when reached) might typically correspond to a total process memory size of 1100 to 1200 MB. Also note that the JVM's memory management algorithms will try to maintain a target steady state heap occupancy of 70% by default, which means that a 1024 MB max heap setting might have a full 300 MB of unused spare memory!

10.1.6 Measuring memory – getting on the same page!

Because there are different ways to measure the memory usage of a process, it is helpful to explain the most familiar metrics and which one is used within this document (and why). Table 10-1 lists and defines the three metrics most commonly used in Windows to measure per-process memory usage, and the different terms (and units) used in Task Manager and Performance Monitor (perfmon) respectively.

The Task Manager terms are the ones used in the column headings in the process page of Task Manager and are measured in kilobytes. The corresponding Performance Monitor counter values are equal to the Task Manager values multiplied by 1024.

All three metrics are routinely measured in the employed methodology, but to calculate the incremental virtual user memory usage, **Private Bytes** [or Virtual Memory Size, in bytes] is the metric adopted for use in this document. This metric is preferred because it is immune to external paging influences, whereas **Working Set** [or Memory Usage, in bytes] can, on occasion, significantly underestimate the process' memory usage in the presence of tight system memory and other conditions that result in paging out of process memory. (To see a dramatic example of this, simply observe this metric for any process with a user interface upon minimizing and restoring its main window.)

Table 10-1 Metrics commonly used to measure per-process memory usage

| Windows Process memory metrics [units] | | Definition |
|--|-----------------------------|--|
| Task Manager [KB] | Performance Monitor [bytes] | |
| VM Size | Private Bytes | Private Bytes is the current size, in bytes, of memory that this process has allocated that cannot be shared with other processes. |
| Memory Usage | Working Set | Working Set is the current size, in bytes, of the Working Set of this process. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from Working Sets. If they are needed, they will then be soft-faulted back into the Working Set before leaving main memory. |
| Peak Memory Usage | Working Set Peak | Working Set Peak is the maximum size, in bytes, of the Working Set of this process at any point in time. |

For the JVM (java.exe), Private Bytes is typically slightly smaller than Working Set (assuming no paging out of java.exe pages is in effect), and Working Set Peak is slightly higher than the maximum value observed for Working Set over time (this is due simply to the fact that the sample interval for Working Set will average out the peak, because the peak is measured for an arbitrarily small unit of time.)

For the measurements collected in creating this document, the differences among these metrics are not discernible at the scale of a typical large virtual user count. Furthermore, because the differences are relatively constant as user load is varied, even at small user counts they don't affect the incremental virtual user memory consumption (that is, slope), just the overhead (that is, intercept).

10.1.7 Methodology for determining the incremental virtual user memory footprint

The key sizing question for load testing is "What is the virtual user memory footprint?" This per-user memory footprint will be referred to as the *incremental virtual user memory footprint*, that is, the memory cost of adding one additional user to the mix. With this value, one can approximate the total memory needs by adding the fixed overhead to the product of the memory footprint per user multiplied by the desired number of virtual users.

The following methodology was used to run tests, collect the measurements, and determine the incremental virtual user memory footprint:

1. To collect memory metrics, two instances of Windows Performance Monitor (perfmon) are used, one sampling every five seconds for 100 samples to monitor the start-up in more detail, and the other sampling every minute to monitor the entire run duration (maximum of 100 samples).

The five process counters collected for the RPT engine process java.exe are:

- % Processor Time
- Elapsed Time
- Private Bytes
- Working Set
- Working Set Peak

The instances are started prior to the run for convenience. They do not collect any data until a java.exe is invoked (because the session java.exe starts first and then exits when the RPT engine starts, the first 10-15 seconds or so of the graph is actually for the session java.exe, but its counters are small and easily ignored).

2. The driver's maximum JVM heap size is constrained, typically to 64 MB, but larger values are used as checkpoints for some workloads. Refer to the second paragraph of “JVM memory heap allocation and garbage collection complexities” on page 327 for this justification.
3. Run durations are set based on the individual test duration, and a sufficient number of test iterations (looping in schedule) are chosen to run at *steady state* for at least 10 to 15 minutes, and typically 20 to 30 minutes. For short duration tests (one to two minutes), steady state is usually achieved before the eight minute perfmon instance wraps around, but each playback is run significantly longer to ensure that the steady state memory consumption has been achieved.

To avoid lock-step and represent reasonable user activity, *paced* loops are usually used in the schedule (*Control the rate of iterations* selected). *Delay before the first iteration of the loop* is selected to most quickly obtain maximum user dispersion over the iteration's duty cycle and thereby accelerate the arrival of steady state). *Randomly vary the delay between iterations* is selected to achieve a Poisson arrival rate for transactions, which represents the ideal model for independent virtual user burstiness.

Other scheduling techniques are used to see what affect that had on memory usage, including non-paced loops with ramp-up and non-paced loops with no ramp-up (that is, lock step).

4. In most cases, the maximum Private Bytes value from the perfmon graph using the one minute sample interval is used as the memory usage value. When this value is unavailable or errant (see the second paragraph of “JVM memory heap allocation and garbage collection complexities” on page 327 as to why any greatly exaggerated value past an established steady state is discounted), the maximum five second interval value is substituted.

Because the counter values are averaged over the sample interval, using a smaller sample interval will result in slightly higher values. The one minute sample interval is preferred due to the nature of the predictive aspect of the methodology, which takes values at smaller user counts to predict memory usage at higher user counts—the general smoothing effect of adding more users is best represented by the larger sample interval at smaller user counts. In the measurements taken for this guide, the choice of the sample interval had only a small impact on the incremental virtual tester memory footprint in any case.

5. Measurements are taken at different virtual user loads to establish a definitive trend, to more easily detect and avoid noise, and to gain confidence in the results. Typically four or five data points are collected for each workload, occasionally more, and always at least three, to get a spread of at least 3-5X from minimum to maximum values.

Most workloads are driven with enough users to start to approach the max heap (but not to get too close, which would distort the linearity of the measurements by eating into the JVM’s 30% spare heap target). The absolute number of virtual users required to achieve both of these goals depended greatly upon the memory footprint; smaller footprints demanded more virtual users to get meaningful differences in the memory usage. User load ranges for various workloads varies from 10 to 50, 100 to 300, and 100 to 1000.

6. When the plotted values for the virtual user load range are available, the intercept and slope are calculated by doing a best fit linear regression of the data points (the Excel® INTERCEPT and SLOPE functions are used). If there is an obviously errant point, that data point is re-run (preferred) or discarded (expedited).
7. The calculated intercept value represents the constant start-up memory footprint of the engine, and is compared against the typical 35 MB value, give or take a few MB for sanity. There can be some legitimate variance in the overhead, such as due to the size of the datapools associated with the test(s) in the schedule.
8. The calculated slope value, which is the ratio of memory use to virtual users, is taken to be the **incremental virtual user memory footprint**.

10.1.8 Possible reasons for overestimating RPT memory usage

It might be useful at this point to mention possible reasons that casual attempts by the uninitiated can result in the perception of per-user memory footprints significantly higher than those provided in this document:

- ▶ **Including workbench memory usage**—If someone adds the workbench and driver engine's process sizes together and divides by the number of users, the per-user memory footprint will be greatly exaggerated. Although the workbench needs to be accounted for in any benchmarking configuration, its memory usage is independent of the number of users in a load test and should be treated as a separate resource, basically a tax for running a load test.
- ▶ **Making a single run and ignoring fixed driver engine overhead**—There is a fixed overhead (approximately 35 MB) for the java.exe driver engine process that is independent of the number of users. If the process size is measured for only a single run and this value is divided by the number of users, the per-user memory footprint will be inflated. This effect is especially pronounced for small user counts and/or workloads which will (otherwise) have small footprints. For example, if a single 50-user run is performed, and the java.exe size is observed at 60 MB, the uninitiated might jump to the conclusion that the per-user footprint is 1.2 MB/user (60 MB/50). Whereas the footprint is likely to be much closer to 0.5 MB/user (60-35)/50). In general, it is bad practice to jump to any conclusions based on a single data point.
- ▶ **Running users in lock-step**—Without getting into a critique of the bad benchmarking practice of synchronizing users such that they perform identical activity in lock-step, it is easy to fall into this trap, because you must take extra steps to un-synchronize user activity to provide more realistic workload dispersion. The steps to stagger user start-up—or better yet to use paced loops with random delays—to spread out the user load to achieve a workload more representative of independent users, are very easy to employ, but can be skipped in a preliminary attempt to check out the memory usage. As will be shown later, running users in lock-step increases the peak http virtual user memory demands. This effect is especially pronounced if the first page has a long response time (such as a login or a big home page) that is exaggerated by the simultaneous requests of many users.
- ▶ **Running a moderately long test with a non-constrained heap and taking a single measurement point post-steady state**—As a general rule it is always best to plot data to establish trends rather than to take an isolated measurement out-of-context, but this is especially true in the context of memory measurements of JVMs. As noted in the second paragraph of “JVM memory heap allocation and garbage collection complexities” on page 327, memory calculations for predictive purposes can be greatly exaggerated by using a post-steady state process size measurement after an over-zealous

JVM memory allocation has occurred. This effect is especially pronounced if the Java heap has not been constrained, because the size of the unconstrained over-zealous allocation can be especially high—in fact, rapid jumps to as high as five times the previously well-established steady state memory usage have been occasionally observed.

- ▶ **Excessive page response times or timeouts**—Until the last request of the page is successfully received, all the page's page element objects remain in memory, therefore either response times that last tens to hundreds of seconds, or page responses with an abnormally high percentage of long response timeouts, will both increase the average memory demands per unit time.

10.1.9 Measuring CPU usage

Measuring CPU usage is for the most part simpler and less controversial than measuring memory. Nonetheless, it might be helpful to explain a few metrics and show you which one is used within this document (and why). Table 10-2 lists and defines three metrics commonly used in Windows to measure CPU usage, and the different terms (and units) used in Task Manager and Performance Monitor (perfmon) respectively.

Table 10-2 Metrics commonly used to measure CPU usage

| Windows CPU Usage Metrics [units] | | Definition and comment |
|--|--|--|
| Task Manager | Performance Monitor (perfmon) | |
| CPU Usage (on Performance tab) [%] | Processor / % Processor Time /_Total (using Object / Counter / Instance hierarchy naming style) [%] | <p>% Processor Time is the percentage of elapsed time that the processor spends to execute a non-idle thread. It is calculated by measuring the duration of time the idle thread is active in the sample interval, and subtracting that time from interval duration. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive, and subtracting that value from 100%.</p> <p>Includes all CPU overhead for the process(es) of interest incurred on all processors, which includes the CPU usage of any background activity not associated with the RPT run, so one should be on the lookout for variations due to unplanned or unknown background activity. Start off with what you believe to be an otherwise idle system. To validate this and to account for background activity you cannot control, it is a good idea to take extended “noise” measurements for durations at least as long as the measurement intervals, as well as to track history over time during the actual measurement interval to detect unexplained variations.</p> |

| Windows CPU Usage Metrics [units] | | Definition and comment |
|--------------------------------------|--|--|
| Task Manager | Performance Monitor (perfmon) | |
| CPU (on Processes tab) [%] | Process / % Processor Time / java (using Object / Counter / Instance hierarchy naming style) [%] | <p>% Processor Time is the percentage of elapsed time that all of the process threads used by the processor to execute instructions. An instruction is the basic unit of execution in a computer, a thread is the object that executes instructions, and a process is the object created when a program is run. Code executed to handle some hardware interrupts and trap conditions are included in this count.</p> <p>Make sure that, if there are multiple java.exe processes running, that the RPT engine's process is selected. This counter has the advantage of excluding activity outside of the RPT engine process (except for some hardware interrupts and trap conditions due to background activity that occur during the execution of instructions belonging to the engine process.) However, be very careful using it on multiprocessors and hyperthreaded processors, because a multi-threaded application (such as the RPT engine) will typically be executing on more than one processor concurrently, and this counter treats the concurrent execution as additive, and the total is not normalized to 100% of the total processing capacity. Therefore non-normalized values taken at face value are exaggerated, which might not be obvious unless the measurement value is greater than 100%.</p> |
| CPU Time (on Processes tab) [secs] | Not available | <p>In Task Manager, the total processor time, in seconds, used by a process since it started.</p> <p>This is the equivalent to the product of the measurement interval and the average of the above % Processor Time counter for the same process during that measurement interval. It is interesting that perfmon has an Elapsed Time counter but no CPU Time counter, whereas Task Manager has a CPU Time counter but no Elapsed Time counter. If both were available on the same tool, one could derive utilization [%] from cumulative CPU time.</p> |

The % Processor Time counter for the processor object (the first metric in Table 10-2 versus the second metric, which is for a specific process) is the one chosen for use in this document when presenting driver CPU usage because it is the easiest and safest to apply, and also represents a more conservative result in practice, as it will tend to overestimate actual RPT engine CPU usage vs. underestimate it. Recall that the driver also consists of the IBM Rational Agent Controller (RAService.exe) process, whose CPU overhead is ignored if the second metric is only applied for the RPT engine's java.exe process. Although this overhead is generally small during the steady state portion of the run (basically proportional to the amount of stats data collection transferred from driver to workbench per unit time), the choice of the first metric naturally includes it, which is appropriate.

Although it is not used as the basis for presenting CPU usage results, the % Processor Time for the RPT engine's java.exe process is also collected as an additional reference when collecting the per-process memory counters.

10.1.10 Methodology for determining the incremental virtual user CPU utilization

CPU usage is collected during a run using the same perfmon instances that are created for measuring memory usage, as outlined in 10.1.7, “Methodology for determining the incremental virtual user memory footprint” on page 329. When we have agreed on how to measure CPU usage, the next step is to define the methodology used to collect the measurements, and generate an answer.

The most common CPU sizing question most load testers will want to ask is, What is the virtual user CPU usage? This allows them to take a workload typically specified with, among other things, the number of virtual users, and approximate the total CPU (processor) needs by taking the fixed overhead (if any) and multiplying the CPU usage per user by the number of users to determine the total number of driver machines needed for CPU processing purposes. We refer to this per-user CPU usage as the *incremental virtual user CPU utilization*, that is, the CPU cost of adding one additional user to the mix. We measure it as a percentage of the processing capacity of a single system (regardless of the number of processors or whether the processors are hyperthreaded or not).

We have chosen the following methodology for this sizing guide:

1. Perform a background CPU check by running the two perfmon instances (five second and one minute intervals) for eight minutes and 30 minutes respectively on the nominally idle driver system. If there is significant CPU activity, its cause should be investigated and *disabled* if possible, in which case the background check should be repeated.

Important: Driver CPU usage measurements should always be made on remote drivers and never on local drivers (local to workbench), because employing this methodology on the latter configuration will result in the inclusion of workbench related CPU activity, and invalidate the driver sizing conclusions.

2. Upon reaching steady state of the applied workload (given that paced loops are used to accelerate arrival at steady state; at minimum, this was after all virtual users have executed the test iteration once, preferably twice), the % Processor / % Processor Time / _Total counter instance is added to the two perfmon instances used to collect per-process memory usage.

3. Prior to leaving steady state (before any virtual user has completed its final iteration), the aforementioned CPU counter is selected, and the average value of the counter for all intervals is read directly from perfmon and used as the single representative value of the CPU utilization for the number of active virtual users. (This average is inclusive of up to the last 100 samples; the intervals of five seconds and one minute are chosen to arrive at an early result within the first 100 shorter samples of one perfmon instance and allow comparison to the final complete result for the entire run duration from the other perfmon instance, which also represents less than 100 samples of the longer sample interval.)
4. The two results are compared as an audit—they should be close because the actual choice of the sample interval does not matter for calculating the average. (The choice of different sample intervals is useful however for looking at the variation in the peaks.) A significant deviation most likely indicates that the shorter initial interval simply has not reached steady state yet, or that some external background activity occurred that discredits the results, the latter of which would require a rerun.
5. Assuming that the audit of the previous step does not require a rerun, the average CPU utilization from the longer perfmon instance is accepted as the legitimate CPU utilization value.
6. Using the same methodology, steps 2 through 5 are repeated for each run at the same varying user loads chosen for memory footprint measurements.
7. When the plotted values for the virtual user load range are available, the intercept and slope are calculated by doing a best fit linear regression of the data points (the Excel INTERCEPT and SLOPE functions are used). If there is an obviously errant point, that data point is re-run (preferred) or discarded (expedited).
8. The calculated intercept value does *not* represent a fixed start-up overhead for the RPT engine, as it does in the case of the memory footprint measurements. Because we are after steady state CPU utilization for driver sizing purposes, we purposely chose a methodology that ignores start-up overhead and only takes measurements during steady-state application of the workload.

However, there is another type of overhead that can show up in the intercept: Any constant background CPU activity that is present in each run that is independent of the virtual user load will show up in the intercept value. For example, there could be a periodic process that runs once every few minutes, which on average consumes 2% of the CPU. (This would normally be detected in step 1 during the background check. In the final sizing calculations, this type of CPU utilization should be ignored if it would not be present on the actual systems when the *official* benchmarks are run, but should be included if it will be present.)

Excluding background CPU activity, one might then think that the intercept should always be zero (in theory, on the surface at least). In practice, however, this is not always the case, because some of the queue overhead processing in the engine is not completely linear with the number of virtual testers, particularly at low user counts, as there is some overhead associated with having activity in the queues independent of the actual size of the queues, and this can vary somewhat by workload. Typically the intercept values are two percent or less, and on a practical level do not affect the sizing outcome.

9. The calculated slope value represents the **incremental virtual user CPU utilization**.

10.2 Driver measurements

Based on the measurement methodologies outlined in the introduction, the results of the memory and CPU usage measurements for the RPT driver are presented in this section, preceded by a description of the machine configurations and workloads measured.

10.2.1 Machine configurations

Table 10-3 lists the machine configurations used for measurements in this chapter.

Table 10-3 Machine configurations used for measurement

| ID | Type | Vendor | Model | Purpose | Processor | Processor Speed | Memory | Operating System |
|----|--------|--------|--------------------------------|------------------------|----------------------|-----------------|---------|---------------------------------|
| A | Laptop | IBM | ThinkPad T42p | Workbench/Local Driver | Single | 2.1 GHz | 2.0 GB | Windows XP SP1 |
| B | Server | IBM | eSeries xServer 235, 86717GX | Remote Driver | Dual, hyper-threaded | 2.793 GHz | 3.8 GB | Windows Server® 2003 Enterprise |
| C | Server | IBM | eSeries xServer 255, 86858BX | Remote Driver | Quad, hyper-threaded | 2.484 GHz | 3.8 GB | Windows Server 2003 Enterprise |
| D | Server | IBM | eServer™ xSeries® 235, 867144X | Remote Driver | Dual, hyper-threaded | 2.384 GHz | 3.75 GB | Windows Server 2003 Enterprise |

10.2.2 Workload descriptions

The salient descriptive data for the workloads measured are shown in Table 10-4. The *Driver Used* column identifies the driver configuration by referencing the ID from the machine configurations table in the previous section.

Table 10-4 Data used for workloads

| ID | Workload | Per-Test Iteration Data | | | | | | Server/ Network Configuration | Driver Used | | |
|----|---------------------------|-------------------------|--------------------|-----------------|-----------------|-----------------------------|------------------|-------------------------------------|----------------|--|--|
| | | # of Pages | # of Page Elements | | | # of Verification Points | | | | | |
| | | | Total | Average Page | Largest Page | Page Title | Response Code | | | | |
| 1 | InternetSearch | 3 | 56 | 18.7 | 47 | 3 | 56 | Internet | A | | |
| 2 | Plants41k | 41 | 306 | 7.5 | 29 | 41 | 306 | Local, 100 megabit | A | | |
| 3 | TradeStatic- WebPages | 5 | 19 | 3.8 | 13 | 5 | 19 | Local, 100 megabit | B,C | | |
| 4 | Internet- ShoppingCart | 7 | 250 | 35.7 | 104 | 7 | 250 | Internet | A | | |
| 5 | TradeSVTApp | 10 | 33 | 3.3 | 12 | 10 | 33 | Local, 100megabit | B | | |
| 6 | Atlantic | 2 | 14 | 7.0 | 7 | 2 | 50 | Local, Gigabit | D | | |

Table 10-5 shows the schedule settings that were used for these workloads (unless noted otherwise in variant measurements).

Table 10-5 Schedule settings used for workloads

| ID | Workload | Schedule Settings | | | | |
|----|----------------------|------------------------|----------------------|-----------------------|-----------------------|-----|
| | | Data Collection Levels | | | Think Time | |
| | | Stats | Execution History | Problem Determination | Mode | Max |
| 1 | InternetSearch | All | Primary Test Actions | Warning | Recorded think times | 10 |
| 2 | Plants41k | All | Primary Test Actions | Warning | Recorded think times | 10 |
| 3 | TradeStaticWebPages | All | Primary Test Actions | Warning | Vary, 50 - 150% range | 10 |
| 4 | InternetShoppingCart | All | Primary Test Actions | Warning | Recorded think times | 10 |
| 5 | TradeSVTApp | All | Primary Test Actions | Warning | Vary, 50 - 150% range | 3 |
| 6 | Atlantic | All | None | None | Recorded think times | 10 |

Note that the data collection levels are out-of-the-box defaults (except for #6), using the out-of-the-box sampling levels as well, which are all users for Stats and Problem Determination, and five fixed users per user group for Execution History.

10.2.3 Memory footprint measurements

For convenience, and consistency with the perfmon graphs, all memory measurements reported in this section are in units of million bytes [mB], which is the natural output of perfmon for the process memory counters. If converted to megabytes [MB], the numbers would be smaller by a factor of $0.953674 = 1000000/1048576$.

Memory footprint for base workloads

Following the methodology set out in the introduction, the incremental virtual user memory footprint was calculated for each of the six base workloads (refer to 10.6, “Intermediate results for memory footprint measurements” on page 369) based on the slope of the perfmon Private Bytes counter for the RPT driver engine process (`java.exe`) as the number of users was varied. The results are shown in Figure 10-3.

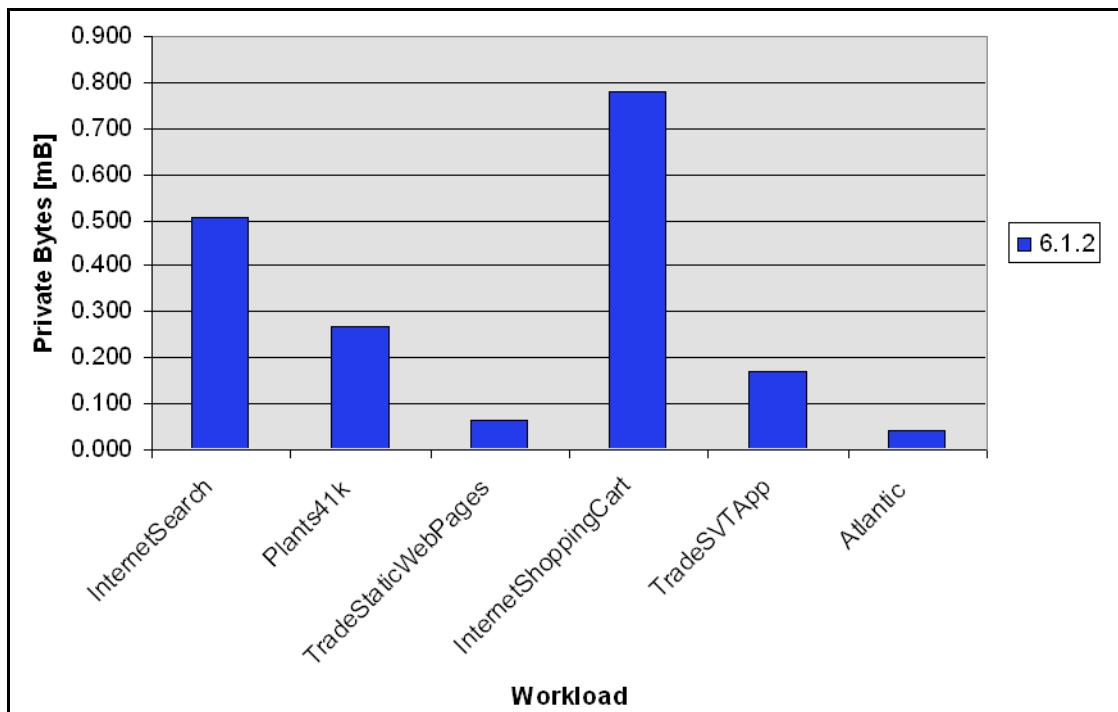


Figure 10-3 Incremental virtual user memory footprint

The actual values graphed in Figure 10-3 are shown in Table 10-6, along with the associated Intercept value. Although the meaning of an average for six disparate workloads is debatable, the average is shown as well, primarily to establish where the *average fixed overhead* of 35 mB quoted elsewhere comes from.

Table 10-6 Incremental virtual user memory footprint

| Workload | Slope [mB] | Intercept [mB] |
|----------------------|--------------|----------------|
| InternetSearch | 0.508 | 24.2 |
| Plants41k | 0.269 | 38.3 |
| TradeStaticWebPages | 0.062 | 34.6 |
| InternetShoppingCart | 0.780 | 34.5 |
| TradeSVTApp | 0.170 | 44.3 |
| Atlantic | 0.043 | n/a |
| Average | 0.305 | 35.2 |

Note that there are two different *clusters* in terms of order of magnitude of the memory footprints for the six workloads. Two workloads are less than 100 KB (where a KB = 0.1024 million bytes), and the other four are between 100 and 1000 KB. There will be more analysis later, but it should be pointed out that the two workloads with the significantly smaller footprints (0.043, 0.062) are for small static Web pages on a local LAN with servers that were able to deliver very small response times. The workloads with the next two larger footprints (0.170, 0.269) were also on the local LAN but were for dynamic applications that generally had larger pages and/or slower relative response times than the previous two workloads. The workloads with the largest footprints (0.508, 0.780) were applications accessed over the Internet where the slower network and yet larger pages had the largest response times.

Measuring the effect of increasing user activity and decreasing workload dispersion

If the relative level of user activity (smaller think times and higher transaction rates) for a given workload is increased, the engine's memory footprint will increase because the average memory demands per unit time will be higher, and (to the extent it occurs) if the server load causes response times to increase, then the fraction of time that objects need to be kept in memory for a page increases as well. Another factor that increases peak memory demands is when user activity is synchronized such that the memory demands of all the users occurs at the same time.

To explore the effect of these factors on the virtual user memory footprint, variations on the base Plants41k and InternetShoppingCart workloads are performed in the next two subsections.

Plants41k variations

The base Plants41k workload (#2) has a 41 page test that nominally takes almost three minutes to execute (174 seconds), and is executed in the schedule using a paced loop at a rate of 15 tests/hour (one test every four minutes), with a random delay before the first iteration to spread out the users. The following variants were run at a 20-user load and compared to the base version:

- ▶ 2a) Instead of using the delay before the first paced loop iteration to spread out the users, each user's start time was staggered two seconds, that is, a *two second ramp*. And instead of using random delays between iterations of the paced loop such that each user ran a test on average every four minutes, the loop was used with no pacing, resulting in the test being run continuously, about once every three minutes (depending upon response time variations).

- ▶ 2b) This variant ran the tests in lock-step. The tests were run continuously with no intervening delay, but more importantly, there were no delays before the first loop iteration and no staggered user start-up. This resulted in every user hitting the first page simultaneously, and the first page of Plants41k was the biggest page (29 elements).
- ▶ 7) A new workload (#7 Plants2k) was created using the first two pages from Plants41k, which happened to be the biggest (29 and 21 elements respectively), raising the average page size by more than a factor of 3.3X. In addition, the test was run continuously with no pacing and also zero think times, thus essentially hitting the server with the two biggest pages with no intervening delays. This essentially creates an extreme worst-case scenario for memory footprint for this application. Table 10-7 compares the workloads of Plants41k and Plants2k.

Table 10-7 Workloads of Plants41k and Plants2k

| ID | Workload | Per-Test Iteration Data | | | | | | Server/ Network Configuration | Driver Used | | |
|----|-----------|-------------------------|--------------------|-----------------|-----------------|-----------------------------|------------------|-------------------------------------|----------------|--|--|
| | | # of Pages | # of Page Elements | | | # of Verification Points | | | | | |
| | | | Total | Average Page | Largest Page | Page Title | Response Code | | | | |
| 2 | Plants41k | 41 | 306 | 7.5 | 29 | 0 | 0 | Local, 100 megabit | A | | |
| 7 | Plants2k | 2 | 50 | 25.0 | 29 | 0 | 0 | Local, 100 megabit | A | | |

Each of the three variations was run at a 20 user load, and the maximum Private Bytes value was observed. Then the incremental virtual user memory footprint was calculated, using the previously calculated intercept for the base Plants41k workload for expediency (rather than re-run each variant for the spectrum of user loads). The results are shown in Table 10-8 and in Figure 10-4.

Table 10-8 Incremental virtual user memory footprint variations: Plants 41K

| Workload | # of users | Private Bytes [mB] | Intercept [mB] | Incremental Virtual User Memory Footprint [mB] | Growth Factor [X] | Description |
|---------------------------|------------|--------------------|----------------|--|-------------------|--|
| 2) Plants41k - Base | 20 | 43.8 | 38.3 | 0.274 | 1.00 | Base workload (paced loop) |
| 2a) Plants41k - Ramp | 20 | 44.3 | 38.3 | 0.299 | 1.09 | 2 second ramp instead of paced loop |
| 2b) Plants41k - Lock-step | 20 | 61.3 | 38.3 | 1.149 | 4.20 | Lock-step (no paced loop or ramp) |
| 7) Plants2k - Zero think | 20 | 73.0 | 38.3 | 1.734 | 6.34 | 2 second ramp, zero think time, 2 biggest pages from Plants41k |

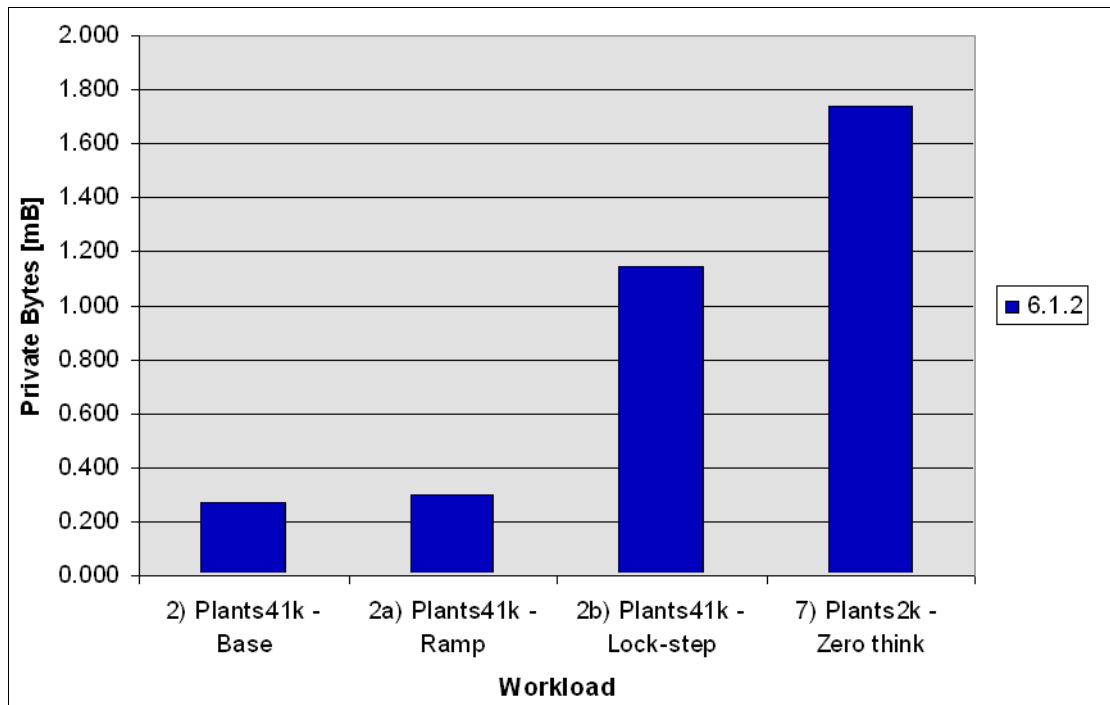


Figure 10-4 Incremental virtual user memory footprint variations: Plants 41k

Note that in variation 2a, increasing the user activity (throughput) of Plants41k by about 1.38X (running the test every 240 seconds on average vs. about 174 seconds depending upon response time variations) and somewhat decreasing the workload dispersion increased the memory footprint only slightly, a little less than 10%.

Variation 2b, which put all the users in lock-step, had a much bigger factor by aligning all the users at the start on the first (and biggest) page, thus creating an initial abnormal peak occurrence for memory demand, and the end result was a footprint increase of over 4X. An examination of the Private Bytes value over time shows that it decreases down to ~54 mB after the initial 61.3 mB peak. But because the employed methodology takes the maximum value of Private Bytes over any interval, the lock-step effect is higher than what an average over time would show.

Variation 7 is an even more dramatic increase because it essentially changes the workload by getting rid of all the smaller pages such that the average memory demand per page is larger, and then increases the frequency of the page memory demands by removing all think times. The resultant memory footprint of over 1.7 mB per virtual user is essentially the worst case for this application. However, one must take into account that the idea of *virtual user* is really being stretched in this case because the continuous activity of each user (with no think time) really represents more than one real user. Also note that with zero think times and continuous activity, the CPU demands of this virtual user will be significantly higher, and that from a sizing perspective, the driver is much more likely to hit a CPU bottleneck than a memory bottleneck in this extreme case.

InternetShoppingCart variations

The base InternetShoppingCart workload (#4) has a 7 page test that nominally takes about 1.25 minutes to execute, and is executed in the schedule using a paced loop at a rate of 20 tests/hour (one test every four minutes), with a random delay before the first iteration to spread out the users over the three minute interval. The following variants were run at a 20 user load and compared to the base version:

- ▶ 4a) The loop rate was increased to 30/hour, so each user started a test on average every two minutes.
- ▶ 4b) The loop rate was increased to 40/hour, so each user started a test on average every 1.5 minutes. In order to reliably achieve this rate, the maximum think time was truncated to 5 seconds.
- ▶ 4c) Loop pacing was turned off and virtual user start-times were not staggered, so that the users ran in lock-step. In addition, think times were set to zero. For the seven page shopping cart scenario recorded in this test, this variant represents the worst case scenario for memory footprint.

- 8) A new workload (#8 InternetShoppingHome) was created using the first page from InternetShoppingCart, the vendor's home page which happened to be the biggest by far (104 elements), raising the average page size by more than a factor of 2.9X. In addition, the test was run continuously with no pacing and also zero think times, thus essentially hitting the server with the biggest page with no intervening delays. This essentially creates an extreme worst-case scenario for memory footprint for this application. Table 10-9 compares the workloads of InternetShoppingCart and InternetShoppingHome.

Table 10-9 Workloads of InternetShoppingCart and InternetShoppingHome

| ID | Workload | Per-Test Iteration Data | | | | | | | Server/ Network Configuration | Driver Used | | |
|----|-----------------------|-------------------------|--------------------|-----------------|-----------------|-----------------------------|------------------|----------|-------------------------------------|----------------|--|--|
| | | # of Pages | # of Page Elements | | | # of Verification Points | | | | | | |
| | | | Total | Average Page | Largest Page | Page Title | Response Code | | | | | |
| 4 | Internet-ShoppingCart | 7 | 250 | 35.7 | 104 | 7 | 250 | Internet | A | | | |
| 8 | Internet-ShoppingHome | 1 | 104 | 104.0 | 104 | 1 | 104 | Internet | A | | | |

Each of the three variations was run at a 20 user load, and the maximum Private Bytes value was observed. Then the incremental virtual user memory footprint was calculated, using the previously calculated Intercept for the base InternetShoppingCart workload for expediency (rather than re-run each variant for the spectrum of user loads). The results are shown in Table 10-10 and in Figure 10-5.

Table 10-10 Incremental virtual user memory variations: InternetShoppingCart

| Workload | # of users | Private Bytes [mB] | Intercept [mB] | Incremental Virtual User Memory Footprint [mB] | Growth Factor [X] | Description |
|------------------------------------|------------|--------------------|----------------|--|-------------------|--|
| 4) InternetShoppingCart - Base | 20 | 50.0 | 34.5 | 0.775 | 1.000 | Base workload (20/hour - one every 3 minutes) |
| 4a) InternetShoppingCart - 30/hour | 20 | 59.9 | 34.5 | 1.270 | 1.639 | 30/hour - one every 2 minutes |
| 4b) InternetShoppingCart - 40/hour | 20 | 62.7 | 34.5 | 1.410 | 1.819 | 40/hour - one every 1.5 minutes (max think of 5 seconds) |

| Workload | # of users | Private Bytes [mB] | Intercept [mB] | Incremental Virtual User Memory Footprint [mB] | Growth Factor [X] | Description |
|---------------------------------------|------------|--------------------|----------------|--|-------------------|--|
| 4c) InternetShoppingCart - Zero think | 20 | 81.5 | 34.5 | 2.350 | 3.032 | lock-step, zero think time |
| 8) InternetShoppingHome - Zero think | 20 | 87.70 | 34.5 | 2.660 | 3.432 | lock-step, zero think time, biggest page from InternetShoppingCart |

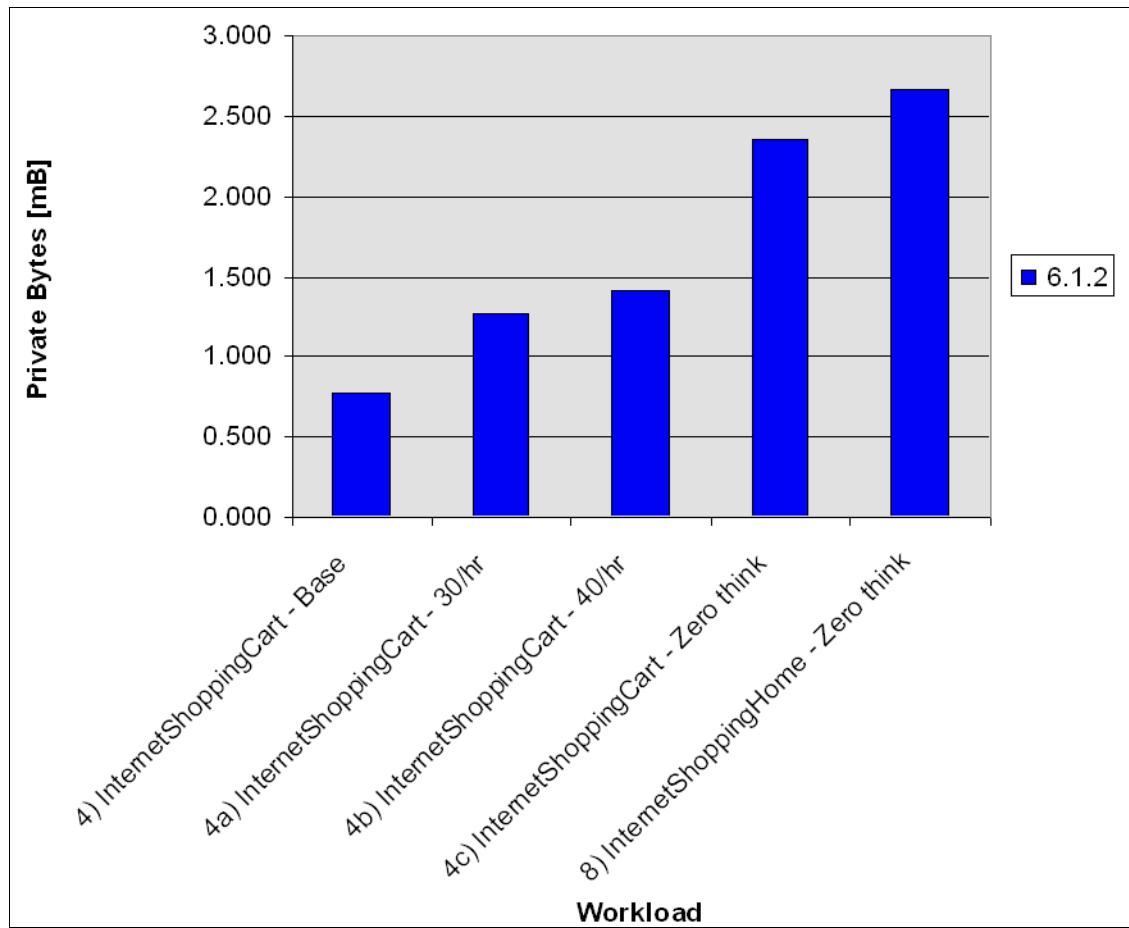


Figure 10-5 Incremental virtual user memory footprint variations: InternetShoppingCart

Note that in variation 4a, increasing the user activity (throughput) by approximately 50% resulted in a memory footprint increase of 64% over the base. This was more than expected, and upon further examination of the data, the following additional factors were uncovered. Unlike Plants41k, the InternetShoppingCart workload is against a live Internet retailer Web site, and the traffic on the server fluctuates based on the Christmas shopping whims of the general public.

The base test was run on a Saturday evening, and the variants were run on a Sunday afternoon. The response times experienced on Sunday were higher than on Saturday afternoon, apparently due to higher site traffic. An additional factor that increased the response times as the throughput was increased was that these particular runs, unlike all the other workloads, were generated from a home cable modem. This modem imposed its own throughput constraints on the response times, which became greater as the throughput was increased.

Variation 4c further increased the virtual user memory footprint up to a factor of 3X over the base workload, up to 2.35 MB. This is high, but again, an extreme case. Compared to a similar extreme in the Plants41k 2b variation, which had a footprint of 1.73 MB, the larger footprint for the InternetShoppingCart 4c variation is attributed primarily to the larger page sizes.

Variation 4d represents not only the extreme case for this Web app, by continuously looping on a very large page (104 page elements) with no delays, but this page size also is towards the high end size wise of reasonable Web pages, so it should represent a situation near the high end of a wide range of Web apps. The same sizing comments (stretched definition of *virtual user*) made regarding Variation 7 of Plants41k apply here as well.

10.2.4 CPU usage measurements

The initial focus of this sizing guide was on memory footprint, so a complete set of CPU usage measurements across all the workloads using the same machine configurations and the same methodology wasn't attempted. For the purposes of this initial version, results will be presented for the three workloads that were run on the dual-processor hyperthreaded server configurations identified with IDs B and D in 10.2.1, "Machine configurations" on page 337.

Summary of CPU usage measurement results

Following the methodology set out in the introduction, the incremental virtual user CPU utilization was calculated for three of the six base workloads (see the Appendix for intermediate results), based on the slope of the perfmon % Processor Time counter as the number of virtual users was varied. The results are shown in Table 10-11, along with additional derived normalized results.

Table 10-11 Summary of CPU usage measurement results

| Workload | Incremental Virtual User CPU Utilization [%] | # of Virtual Users @ 70% CPU | # of Page Elements / Iteration | Iteration Cycle Time [sec] | Average Hits / Sec / Virtual User | Incremental Virtual User CPU Utilization (Throughput Normalized @ 1 hit/sec) | # of Virtual Users @ 70% CPU (Throughput Normalized @ 1 hit/sec) | Machine Configuration ID | Processor Speed [GHz] | # of Processors | (Processor and Throughput Normalized) CPU Util of 1GB single processor / 1 hit/sec [%] |
|----------------------|--|------------------------------|--------------------------------|----------------------------|-----------------------------------|--|--|--------------------------|-----------------------|-----------------|--|
| TradeStatic-WebPages | 0.075 | 933 | 19 | 30 | 0.633 | 0.118 | 591 | B | 2.793 | 2 | 0.662 |
| TradeSVT-App | 0.175 | 400 | 33 | 60 | 0.550 | 0.318 | 220 | B | 2.793 | 2 | 1.777 |
| Atlantic | 0.0422 | 1659 | 14.0 | 60 | 0.233 | 0.181 | 387 | D | 2.384 | 2 | 0.862 |

There is a range of over a factor of four in the CPU usage for the three workloads, which corresponds to supporting from a low of 400 to a high of nearly 1700 virtual users per dual-processor driver machine. When the workloads are normalized to the same hit rate, the range is reduced to less than a factor of three between the three workloads. This indicates, like memory, that the number of virtual users that can be supported per driver machine before being limited by CPU processing capacity might vary significantly for different workloads (just based on this small sample size) even after accounting for obvious differences in http request throughput.

Subsequent sections explain the various columns and provide graphs where useful.

Incremental virtual user CPU utilization

The incremental virtual user CPU utilization values from the second column of Table 10-11 are shown graphically in Figure 10-6.

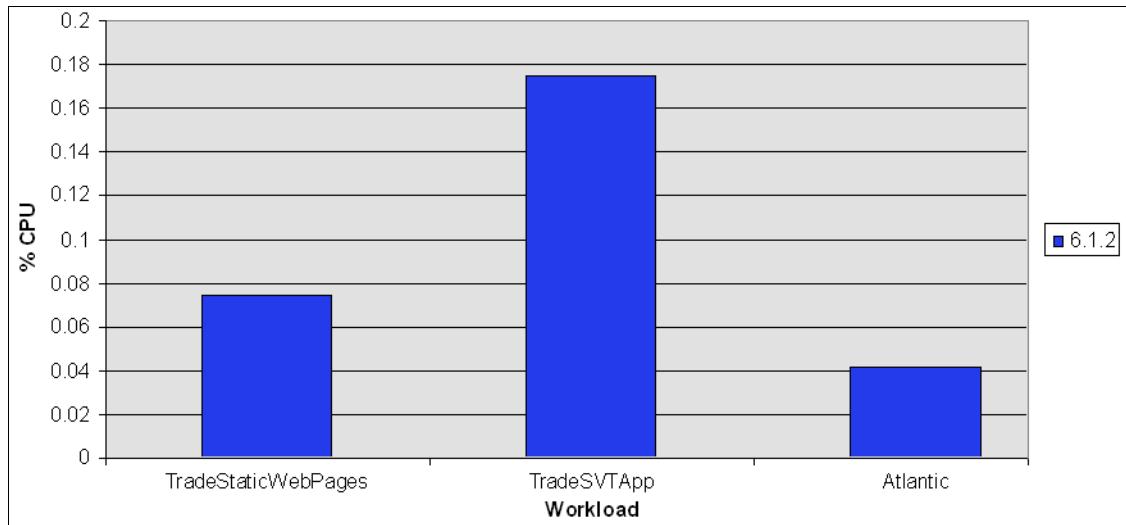


Figure 10-6 Incremental virtual user CPU utilization

Number of virtual users @ 70% CPU

Using the guidance and general rule-of-thumb of not using more than 70% CPU on average on the driver machine, the maximum number of virtual users that can be supported on the driver is calculated by dividing 70 by the incremental virtual user CPU utilization. The results of column 3 of Table 10-11 are shown in Figure 10-7.

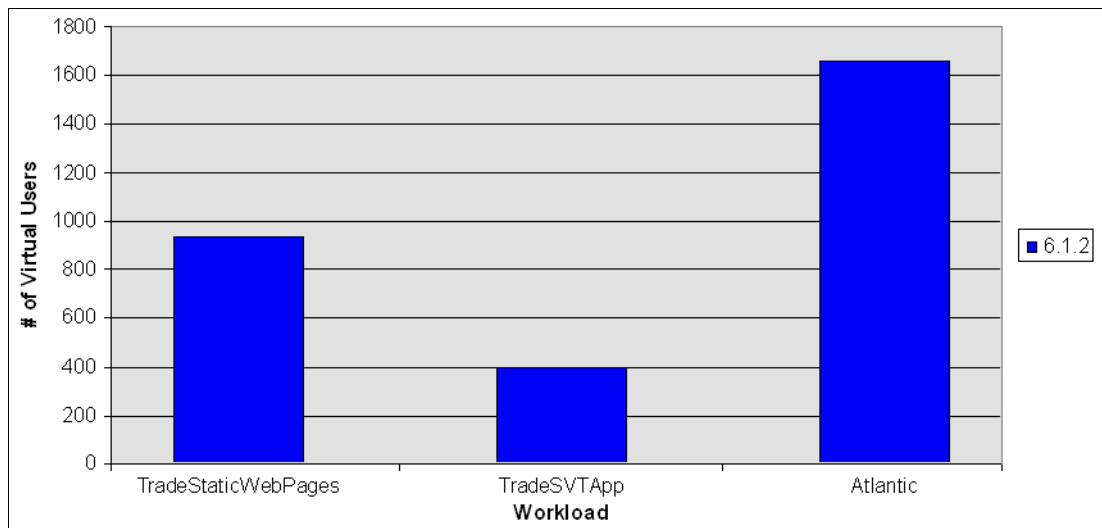


Figure 10-7 Number of virtual users at 70% CPU

Average hits / sec / virtual user

This is not a CPU measurement *per se*, but it is a workload metric very relevant to RPT HTTP CPU usage. Most of the processing associated with an HTTP test is incurred on each HTTP request (one for each page element within a page). It can be calculated by dividing the number of page elements per test iteration (column 4) by the average iteration cycle time (column 5). The results are shown in column 6, and will be used to normalize the CPU utilization, as described in the next section.

Incremental virtual user CPU utilization (normalized)

The incremental virtual user CPU utilization values from the second column are normalized to a throughput of 1 hit/second by dividing by the corresponding average hits / second / virtual user for each workload (column 6), and the results are shown in column 7 and graphed in Figure 10-8.

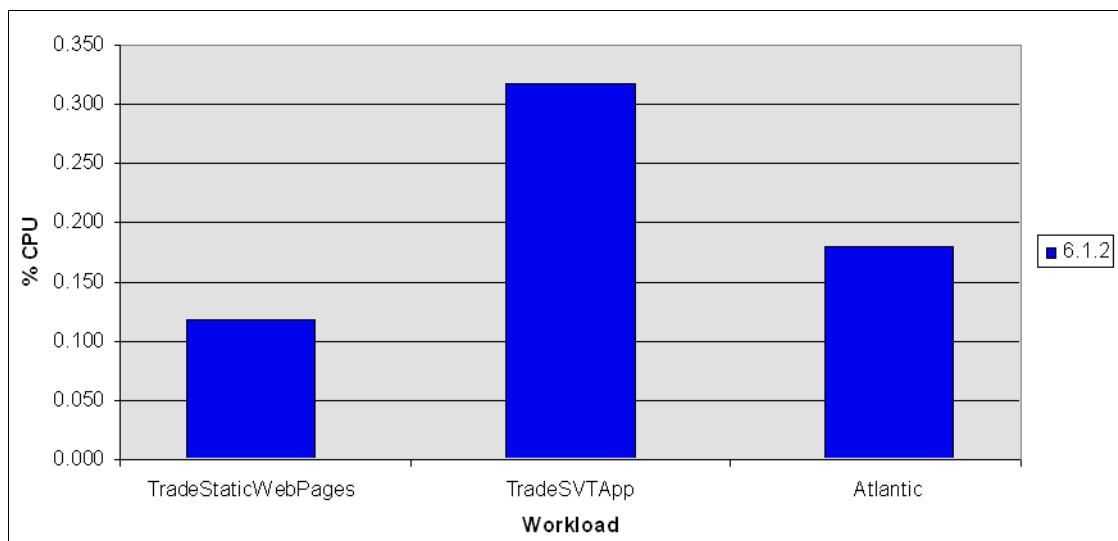


Figure 10-8 Incremental virtual user CPU utilization (throughput normalized @ 1 hit/sec/user)

Although normalizing the throughput reduces the variation between workloads, significant differences in CPU usage remain. The processing overhead per http request can vary significantly based on the size of the response, and whether additional data correlation processing or verification point processing is required on the response. (Data correlation processing involves regular expression matching on the response content to locate and extract strings that must later be substituted into the content of subsequent http requests by the same virtual user.)

Number of virtual users @ 70% CPU (normalized)

Using the guidance and general rule-of-thumb of not using more than 70% CPU on average on the driver machine, the maximum number of virtual users that can be supported on the driver at a throughput of 1 hit/sec is calculated by dividing 70 by the normalized incremental virtual user CPU utilization (column 7). The results are shown in column 8 and are graphed in Figure 10-9.

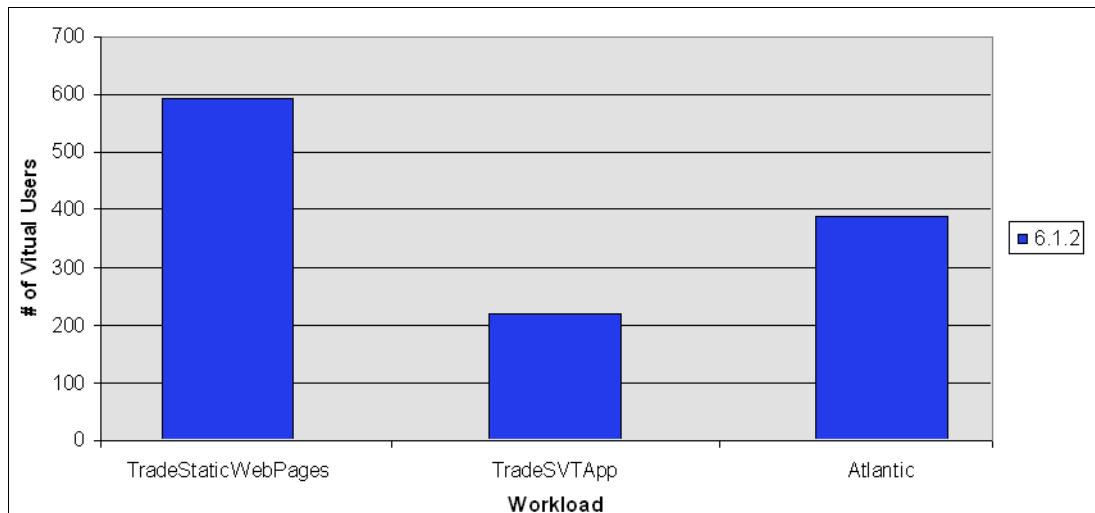


Figure 10-9 Number of virtual users at 70% CPU (throughput normalized @ 1 hit/sec/user)

10.3 RPT sizing guidelines

In this section we provide guidelines for sizing of RPT.

10.3.1 Overview and approach

The general approach to sizing the hardware requirements for a load testing engagement depends upon how much workload information is actually available and whether preliminary access to the application under test is possible. The approach also depends on the purpose and constraints of the sizing exercise: How much accuracy is desired and what is the tolerance for overestimating (budget or procurement concerns) or underestimating (can you go back to the source to procure additional hardware if needed?).

If you have no access to application recordings or workload, you are mostly “flying blind”, and the recommended guidance is to use initial estimates of 1 MB memory per virtual user and a CPU capacity of 500 virtual users for a dedicated

driver machine using a reasonably configured dual processor server (2.5-3 GHz with at least 2 GB memory), with an assumption that there is an ability to add additional driver machines if needed. However, if all the hardware has to be procured up front and there is very low tolerance for coming up short, we recommend following a more conservative approach of allowing 2 MB memory per virtual user and reducing the CPU capacity in half to 250 virtual users. Although this will likely result in the acquisition of excess hardware not needed, it is the prudent course if there is no ability to acquire additional hardware later.

If you do have workload information but don't have access to the application and an ability to execute a few sample trial runs, then you should consult the guidelines in "Memory" on page 355 to more accurately predict memory needs and the guidelines in "CPU" on page 355 to more accurately predict CPU needs.

Otherwise, for the most reliable sizing guidance, we recommend obtaining some sample scenario recordings for the application(s) to be load tested and perform memory sizing projections with a single laptop or desktop machine as follows:

- ▶ Constrain the JVM heap to 64 MB.
- ▶ Follow the methodology for determining incremental virtual user memory footprint (Section 10.2.6) for 10, 30, 50, 70, and 100 users. If necessary either increase the JVM heap or reduce virtual user counts, but at least get three, and preferably five different data points by which to calculate the incremental virtual user memory footprint for the actual workload scenarios. From this data, you can follow the guidance in Section 10.4.4.1.3 to determine how many virtual users can be handled per driver machine from a memory needs perspective (that is, if the RPT engine's virtual user capacity is memory bound).

For CPU sizing, it is best to utilize a representative dedicated driver machine, but if that is not possible, make the measurements using another available machine and pro-rate the measured CPU capacity results by accounting for the CPU processing differences between the trial machine and the actual machine to be acquired for the load testing project. Follow the CPU measurement methodology described in 10.1.10, "Methodology for determining the incremental virtual user CPU utilization" on page 335.

Avoid small runs that use less than 10% CPU. Try to get CPU results for at least three runs in the 10-30% range, and preferably 10-60% range. From those results, project how many virtual users can be driven at 70% CPU utilization, assuming the RPT engine's capacity is CPU bound. Note that for most workloads, as of RPT 6.1.2, RPT tends to be CPU bound for typical workloads and common machine configurations of processing power and memory. But always examine both the projected memory and CPU needs to guide the process of selecting machine configurations.

10.3.2 System configuration

There are two possible configurations for running a load test with RPT. These are described in the following sections, along with system tuning considerations.

Dedicated workbench console and remote drivers

This is always the recommended configuration for any serious load testing or benchmarking application. The workbench is dedicated to console activity and dedicated remote drivers are used to generate all of the virtual user load. It has many advantages and should be considered a standard best practice.

When you have a choice, it is better to utilize fewer faster driver systems than a larger number of smaller driver systems. Fewer drivers means less workbench; driver communication overhead and the run startup and cleanup times are shorter. The majority of the driver initialization is done in serially, which can lead to lengthy startup times for runs involving tens of drivers. There are also some additional memory savings on the workbench with fewer driver systems; normally these savings are small, but if you choose to keep individual statistical data for each driver machine (not the default) then the stats model memory used to store report counters grows proportional to $D+1$, where D is the number of drivers.

Workbench console and local driver (single machine)

This configuration is not recommended for any serious load testing or benchmarking effort. However, there are a few cases where its disadvantages are either inconsequential or tolerable, and necessity or convenience might dictate its use: test development or debugging with very low virtual user counts, typically 10 or less; load testing with a very light workload under 50 or perhaps 100 virtual users; or load testing when the only purpose is to generate a load without regard to response time measurements and the desired load is demonstrated to be within the capacity of the workbench machine alone.

Another situation where it might be tolerable to generate virtual user load from the workbench console machine is if you are using your full complement of remote driver machines and are severely pressed for additional driver capacity, but you should only consider this if the workbench console is running on a large multi-processor workbench machine that has more than ample reserve CPU and memory capacity to provide the remaining load.

Although not recommended, if you must resort to driving virtual user workload from the workbench console machine, it is important to follow these practices to minimize potential performance problems:

- ▶ Close all project assets (other than the minimal necessary reports on the current run) before schedule execution, and do not open project assets (especially large ones) until after the schedule execution has completed.

- ▶ During the run, avoid unnecessary operations on the workbench console, and especially avoid any CPU intensive operations, even of relatively short duration.
- ▶ Appropriately balance workbench and local driver JVM heap memory allocation.
- ▶ If you have less than 4 GB on the workbench machine, always use a location for the local driver and constrain its heap so that you do not allow the workbench machine to use more memory than physically available, which will result in poor performance due to paging overhead and thus compromise measurement results.

10.3.3 Workbench sizing

The primary scope of this section is the driver system and not the workbench. This is because utilization of workbench resources is largely unaffected by the number of virtual users and workload characteristics, whereas the driver system sizing is heavily influenced by those factors, and therefore driver system sizing determines how much total hardware is needed to generate the desired load. Accordingly, we provide a few basic workbench sizing guidelines and tips, and refer you to *Rational Performance Tester 6.1.2 Workbench Memory Optimization* (IBM technote 1221972) for details.

Memory

Although the workbench memory needs are basically independent of the number of virtual users, the workbench can demand high amounts of memory for opening and viewing large project assets, especially large report result sets and test logs. Therefore the amount of data that can be collected and viewed is limited by the amount of memory available on the workbench machine, and more specifically limited to the maximum size of the Java heap allocated to the RPT Eclipse workbench process.

For most serious load testing purposes we recommend 2 GB memory on the workbench machine, primarily to maximize the data collection capacity and improve responsiveness. Additional memory is not generally advantageous (at least not for RPT purposes). If you have 2 GB memory or more, you can optimize RPT performance by setting the max heap to 1200 MB (up to 1500 MB if possible—this is machine configuration dependent.)

If you are interested in detailed aspects of memory usage and optimization related to the workbench, refer to the related document, *Rational Performance Tester 6.1.2 Workbench Memory Optimization* (IBM technote 1221972), which was introduced at the beginning of this chapter (see “Rational Performance Tester 6.1.2 Workbench Memory Optimization” on page 321).

CPU

Assuming that you are using the strongly recommended configuration where all virtual user load is generated by remote driver machines, the workbench CPU does not impact the ability to apply load. As such, you can generate very large loads with a slow workbench CPU by exclusively relying on remote driver systems for that purpose. However, there are many other aspects of using RPT ranging from test authoring to result analysis, and they all benefit from a faster CPU. A dual processor configuration is normally not important, but it can be useful for:

- ▶ Very long runs, because the EMF model load time increases with model size.
- ▶ Better responsiveness for user interactivity during a run.
- ▶ Use of RPT's response time breakdown functionality. Unlike the test log, the response time breakdown data is loaded into the workbench memory during the run.

10.3.4 Driver sizing

Driver system sizing is heavily influenced by the number of virtual users and workload characteristics. As the size of the workload increases, the number and/or size of driver systems must be increased accordingly, and therefore driver system sizing is the dominant factor in determining how much total hardware is needed to generate the desired workload. The two primary considerations for driver system sizing are memory and CPU.

Memory

In this section we discuss considerations regarding memory usage.

Factors that influence driver memory usage

Memory usage on the driver machine relates mostly to the workload size and its characteristics. For any given workload, memory usage is mostly proportional to the number of virtual users—when you have accounted for the fixed overhead of the Rational Agent Controller (about 20 MB; see 10.1.4, “RPT driver architecture” on page 325) and the base memory usage for the java.exe driver engine process (about 35 MB, see 10.2.3, “Memory footprint measurements” on page 339) that is independent of the number of users.

The factors that influence driver memory usage changed dramatically beginning with release 6.1.2 (for a comparison with the previous 6.1.1 release, see 10.5.3, “Comparison of Plants41k driver memory usage for RPT 6.1.2 versus RPT 6.1.1” on page 367.)

Following are five workload factors that increase memory usage significantly:

- ▶ Many page elements, that is, HTTP requests per page—Page element storage exists for the entire duration of the page.
- ▶ Small think times—High user activity increases the *density* of the memory demands because the fraction of time the memory required to process each page is utilized (not available for other uses) increases when there is less idle time between transactions.
- ▶ Virtual users that run in lock-step—Low *dispersion* aligns the memory demands of the individual virtual users, leading to higher memory *peaks*, forcing the JVM to allocate more process memory to accommodate these demands.
- ▶ Large post data sent to the server and/or large responses such as for large images received from the server—High I/O requires more memory for buffer space.
- ▶ Long page response times—The memory associated with the page and each of its page elements is held longer.

Next we describe the quantitative impact of these factors on memory footprint.

Summary guidelines for driver memory

The following summary guidance for driver memory footprint has been established in Table 10-11 based on the data collected for the six workloads analyzed in 10.2.3, “Memory footprint measurements” on page 339.

Table 10-12 Summary guidelines for driver memory

| Incremental Virtual User Memory Footprint (per virtual user) | Workload Characteristics |
|---|--|
| 40 - 100 KB | Small pages, fast response times (typically local server), light to medium activity, medium to high dispersion. |
| 250 KB - 1 MB | Medium to large pages, fast to medium response times (typically internet), medium activity, medium to high dispersion. |
| 2 - 3 MB | Large pages, slow response times, high to extreme activity (zero think times), low dispersion (lock-step). |

Estimating a driver's virtual user capacity based on memory usage

Regardless of the method used, when you have estimated the incremental virtual user memory footprint, you can then estimate the virtual user capacity for a particular driver that is memory bound based on its memory configuration.

This might seem like a trivial task: Divide the available memory by the incremental virtual user memory footprint. However, what value should be used for the *available memory*? Recall that the incremental virtual user memory footprint is related to the growth of the java.exe driver engine process size. Although the largest part of the process size is due to the JVM heap, process memory is also consumed for the native code implementation of the JVM itself which consists of its code space, its stack space, and its own heap.

So if you use the allocated JVM maximum heap value for *available memory*, you will underestimate the potential number of virtual users because you are ignoring the engine's java.exe process overhead component already included in the calculation of the incremental virtual user memory footprint. But if you use the machine's physical memory, you will overestimate the potential number of virtual users, because you are ignoring the memory consumed by the operating system and other running processes (including the Rational Agent Controller), as well as the base overhead of the java.exe driver engine process (which is not included in the incremental virtual user memory footprint.)

The recommendation is to use the values for available memory from Table 10-12, which should be sufficient for initial sizing purposes. They were based on measurements observed on Windows XP and also factor in the average 35 MB base memory usage for the java.exe driver process. Note that the percentage of memory available for Virtual Users increases with available physical memory because the overhead (while not fixed) does not grow linearly with the amount of physical memory.

Table 10-13 Available memory for virtual users vs. physical memory

| Driver Machine's Physical Memory | Available Memory for Virtual Users |
|----------------------------------|------------------------------------|
| 512 MB | 375 MB |
| 768 MB | 605 MB |
| 1024 MB | 845 MB |
| 2048 MB or more | 1500 MB |

For example, if the estimated incremental virtual user memory footprint is 750 KB, the virtual user capacity on a 512 MB driver machine that is memory bound would be 500 virtual users ($375 / 0.75$). On a 1024 MB machine, the virtual user capacity would be 1126 virtual users ($845 / 0.75$). On a 2 GB or larger machine, the virtual user capacity would be 2000 virtual users ($1500 / 0.75$).

It is important to realize that this estimation of virtual user capacity is based on memory usage considerations alone. It is possible, and in fact likely, that CPU usage will be the initial performance bottleneck rather than memory and hence the virtual user capacity will be CPU bound, meaning that the memory bound calculations in this section will represent upper limits that are in fact not achieved in practice.

CPU

In this section we discuss considerations regarding CPU usage.

Factors influencing CPU usage

CPU usage on the driver machine relates mostly to the workload size and its characteristics. For any given workload, memory usage is mostly proportional to the number of virtual users.

Eight workload factors that increase CPU usage significantly are listed below. The first three factors in combination dictate the http request rate for each virtual user. The remaining factors influence the amount of processing that is required for each http request. The product of the number of http requests per second and the average amount of CPU overhead to process each request determine the incremental virtual user CPU utilization:

- ▶ Many page elements per page—The processing per page is roughly proportional to the number of page elements per page, all else being equal.
- ▶ Small think times—High user activity increases the *density* of the CPU demands, because the fraction of time the CPU is busy processing each page increases when there is less idle time between pages.
- ▶ Short page response times—For the same reasons as stated before, for small think times.
- ▶ Large post data sent to the server or large responses (such as images) received from the server—High I/O requires more processing overhead per request, all else being equal.
- ▶ Frequent data correlations—Data correlation processing involves regular expression matching on the response content to locate and extract strings that must later be substituted into the content of subsequent http requests issued by the same virtual user. The more correlations that exist and the larger the response the regular expression(s) must be matched against, the more CPU time is consumed for data correlation processing.
- ▶ Verbose log levels and sampling rates for test logs and problem determination—Turning up logging levels can add significant processing overhead per request. Of course, when trying to run large numbers of users, you should set the logging levels to the bare minimum you can get by with.

- ▶ Number and type of verification of points—Verification points require processing overhead, which is typically small for page title, response time code and size verification points, but can be large for content verification points.
- ▶ Looping in a schedule vs. looping in a test—Looping in a test is more efficient and allows re-use of HTTP connections across loops—otherwise the default is to close connections at the end of the test. The efficiency gains are very pronounced for short tests containing only a small number of requests. Note that is perfectly fine to loop in a schedule when the test(s) represent reasonably sized user sessions, consisting of many transactions or longer than five minutes; otherwise you should always consider optimizing driver capacity by looping in the test.

Estimating a driver's virtual user capacity based on CPU usage

Regardless of the method used, when you have estimated the incremental virtual user CPU utilization for a representative driver machine, one can estimate the virtual user capacity for a particular driver that is CPU bound.

Using the guidance and general rule-of-thumb of not using more than 70% CPU on average on the driver machine, the maximum number of virtual users that can be supported on the driver is calculated by dividing 70 by the incremental virtual user CPU utilization (expressed as a percentage). Examples of this calculation were made for three different workloads in Section 10.3.4.

It is important to realize that this estimation of virtual user capacity is based on CPU usage considerations alone. It is possible that memory usage will be the initial performance bottleneck rather than CPU, and hence the virtual user capacity will be memory bound, meaning the CPU bound calculations in this section will represent upper limits that are in fact not achieved in practice.

Summary guidelines for driver CPU

For convenience, the driver virtual user capacity results from Section 10.3.4 are consolidated in the first two rows of Table 10-14 with the salient configuration information. To facilitate the analysis and simplification of these results, a third row was added that represents a fictitious 2.5 GHz version of the same machine model, and its results were projected by applying a linear scaling of the results based on the respective clock speeds of the other two configurations. (This is a safe, accurate projection because both configurations were the same model and architecture, and the clock speed of the *new* configuration only varied by a small fraction of the clock speed of the original two configurations.)

Table 10-14 Consolidation of driver CPU usage memory results

| Server Hardware | CPU Speed | Main Memory | Virtual User Capacity by Workload | | | | | |
|-------------------------------------|-------------------------|-------------|-----------------------------------|-----|------|---------------------------------|-----|-----|
| | | | As Measured | | | Normalized to 1 hit/second/user | | |
| | | | #3 | #5 | #6 | #3 | #5 | #6 |
| IBM eSeries xServer 235 (2 CPUs) | 2.793 GHz | 3.8 GB | 933 | 400 | n/a | 591 | 220 | n/a |
| | 2.384 GHz | 3.75 GB | n/a | n/a | 1659 | n/a | n/a | 387 |
| | 2.5 GHz (normalized) | 3.8 GB | 835 | 358 | 1740 | 529 | 197 | 406 |

We can now summarize in Table 10-15 the range (low and high) and average virtual user capacity for the normalized 2.5 GHz driver machine for both the original *as measured* workloads and the normalized (to an HTTP request rate of 1 Hit/Second/User) versions of the same workloads.

Table 10-15 Simplified summary of driver CPU usage memory results

| Server Hardware | CPU Speed | Main Memory | Virtual User Capacity (rounded to 10 VU) | | | | | |
|-------------------------------------|-------------------------|-------------|--|------|-----|---------------------------------|------|-----|
| | | | As Measured | | | Normalized to 1 hit/second/user | | |
| | | | Low | High | Avg | Low | High | Avg |
| IBM eSeries xServer 235 (2 CPUs) | 2.5 GHz (normalized) | 3.8 GB | 360 | 1740 | 980 | 200 | 530 | 380 |

Even after normalizing out the differences in HTTP request rates, there is still a wide variation in virtual user capacity of a factor of 2.6 for the three workloads. Without normalization, the variation is much higher: a factor of 4.8. Given that this was a relatively small sampling of workloads, and that large variations in several of the factors influencing the per-request overhead were not present, these results do not represent the full variability that would be expected over the range of workloads that might be encountered in the field.

Therefore, when needing to accurately project the virtual user capacity of a driver, it should be re-emphasized how important it is to record actual application scenarios and perform some trial runs to measure the incremental virtual user CPU utilization. Refer to the final paragraph on CPU sizing in 10.3.1, “Overview and approach” on page 351 for guidance on how to perform those trial runs.

However, if it is not possible to make trial measurements, or if you need an initial sizing estimate that will be refined later with actual measurements, and you have at least some information about your anticipated workload and hardware resources, here is what we recommend. Use the consolidation of the driver virtual user capacity results measured in 10.2.4, “CPU usage measurements” on page 347 as a starting point, and tailor those results to target your workload and hardware based on the following considerations:

- ▶ Compare your available hardware to that used for the CPU measurements in 10.2.4, “CPU usage measurements” on page 347 and scale up or down appropriately based on relative CPU processing power. Scaling solely on CPU clock rates between dissimilar models can be very misleading due to variations in primary, secondary, and L3 cache sizes, memory access time, bus speeds, and the chip/core/hyper-threading architecture, each of which will impact effective processing throughput.

Performance also typically does not scale linearly with the number of processors due to bus and memory contention. A more reliable way to compare relative CPU processing power for the RPT load testing engine application is to use the results of industry standard benchmarks that are tailored to integer and string processing throughput for multi-processing systems, preferably in Java, such as:

- Standard Performance Evaluation Corporation (SPEC)'s JBB2005:
<http://www.spec.org/jbb2005/results/jbb2005.html>
- CINT2000 Rates:
http://www.spec.org/cpu2000/results/cpu2000.html#SPECint_rate
- CINT2006 Rates:
<http://www.spec.org/cpu2006/results/rint2006.html>

The latter two are not Java based, but are still useful and benefit from a much larger set of published results. Avoid floating point benchmarks and benchmarks that measure single-copy speed as opposed to multi-copy throughput, because RPT is multi-threaded and its architecture is fully capable of taking advantage of multi-processor systems (at least up to 32 processors with the default settings).

- ▶ Compare your anticipated HTTP request rate (not the page rate or transaction rate, but the individual HTTP request rate) to the normalized 1 hit/second/user results, choosing the results from the measured workload that best matches your anticipated workload (see 10.3.2, “System configuration” on page 353 for workload descriptions), or use the average of the three workloads if you are totally unsure. If you know of sizing results for a previous workload that is very similar and differs primarily in the HTTP request rate, that is probably the best reference source.

For estimation purposes it is appropriate to scale linearly based on the HTTP request rate. For example, if the anticipated request rate is 1.5 hits/second/user then you should expect to support 1.5 times the number of users as compared to the results normalized to 1 hit/second/user.

- ▶ Make any additional adjustments up or down based on the other factors influencing CPU usage outlined in “Factors influencing CPU usage” on page 358 if you believe some of them are very relevant to your workload and not present in the base workload.

Finally, if you have no workload information at all and do not have access to the application under test, refer to the guidance in 10.3.1, “Overview and approach” on page 351 on how to proceed with default sizing assumptions.

Measurement accuracy as a factor in driver CPU sizing

The general guideline to maintain reasonably accurate measurements is to stay below 70% average CPU for sample sizes in the 15 - 60 second range. As average CPU utilization exceeds 70%, the queuing delays will begin to materially impact response time measurements. The measurement error is independent of the response time values themselves, so percentage-wise the impact will be much greater on small response times (a few tenths of a second or less) than medium response times (one to five seconds) and likely will have very little affect on large response times (10 seconds or more). Also, the measurement error will have a higher percentage impact on the accuracy of the upper end of the response time distribution (for example, 90th percentile, 95th percentile) than on the average response time.

Therefore, for situations that demand high response time measurement accuracy and are most affected by potential measurement error (for example, if you have small response times and must validate requirements on the 95th percentile), you should be more sensitive to CPU load on the drivers and might want to back off the 70% guideline to 60% or even 50% to be conservative. Of course, such an approach will require more hardware than might actually be needed to achieve your accuracy requirements, in which case the following measurement-based approach can be useful.

The following technique can be used to assess and monitor the relative response time inaccuracies caused by heavily loaded drivers:

1. Use an additional lightly loaded *reference* driver machine. Assign enough virtual users on the reference driver to obtain statistically meaningful samples, but not too many to create significant load on the driver machine. The goal should be to keep the driver's average CPU utilization at 10% or less, so the measurements taken by this lightly loaded driver are not adversely affected by CPU load, and can serve as the reference against which measurements taken by the more heavily loaded driver machines can be compared.

2. Run the schedule with the *Only store All Hosts statistics* check box cleared. (Note that this will increase workbench memory demands for the RPT report stats counters nearly proportional to the number of drivers used, so if you are close to the workbench JVM max heap limit, you might need to shorten the run duration or not run with the full complement of drivers each run.)
3. Run response time reports separately for each driver (right-click on driver's node, a child of the run of interest).
4. Compare response time statistics of the heavily loaded drivers against the reference driver.
5. If the error is not within acceptable tolerance, re-balance the number of users by decreasing the number of virtual users assigned to the offending driver(s), and shift that load to more lightly loaded drivers, adding new driver machines as needed.

If you have a sufficient number of identical driver machines, you can speed up this process by varying the number of virtual users assigned to each driver machine in the same run to gather multiple data points in parallel, as a function of the virtual user load and CPU utilization on each driver machine.

Additional driver sizing examples

In addition to the virtual user capacity results for the three workloads on the dual processor server machines reported in 10.3.3, “Workbench sizing” on page 35410.3.4, Table 10-16 provides additional examples of what is achievable for light-to-medium workloads on a few different hardware configurations, taking both memory and CPU into account. Unfortunately, information is not available to quantitatively compare these workloads to the three workloads for which CPU usage was measured in this chapter.

Table 10-16 Additional driver sizing examples

| Server Hardware | CPU Speed | Memory | Virtual User Capacity |
|----------------------|-------------------|--------|-----------------------|
| xSeries 330 (2 CPUs) | 800 MHz (PIII) | 1 GB | 800 |
| xSeries 346 (2 CPUs) | 3.4 GHz Dual Core | 3.2 GB | 1500 |
| AIX JS21 (2 CPUs) | 2.7 GHz (64-bit) | 16 GB | 2500 |

As stated elsewhere, results are very sensitive to workload characteristics, and these additional examples are offered to provide more data points. Your mileage might vary, especially if you accelerate *per user* activity, thus increasing the HTTP request rate into a medium-to-high workload range. Do not use these examples for sizing purposes as a shortcut to the approach recommended in 10.3.1, “Overview and approach” on page 351. Even if you have no workload information at all, you should first consult the more conservative guidance provided in that section.

10.4 Setting driver maximum JVM heap size

As discussed in “JVM heap limits” on page 326, the value of the maximum JVM heap size determines how much memory is actually available to the driver’s execution engine, and is therefore a critical factor in RPT scalability. This section provides information on how to tune this parameter by overriding RPT’s default’s heap size algorithms. This must be done on a per driver basis.

10.4.1 Remote driver

For each location specified in a schedule user group, RPT creates a TPTP location asset. RPT_VMARGS is the property name on the location which RPT uses to let the end user supply that driver engine’s JVM command line arguments. To set the maximum JVM heap value, specify a property value of `-Xmx1024m` (for example). The `-Xmx` is the JVM max heap argument. The `1024m` specifies 1024 megabytes (1 GB) of max heap. This only works properly if the system has at least that much, but preferably more real RAM than the value specified. Do not use more than the maximum supported value of the JVM.

For the IBM 1.4.2 Sovereign JVM used by the RPT 6.1.2 engine, we do not recommend exceeding 1900 MB (the Linux limit), although—if it is really needed—on a Windows machine with more than 2 GB of memory, you might be able to use a value as high as 2000 MB successfully. (As of RPT 7.0 the version of the IBM JVM has been updated to 1.5 and the practical maximum heap value is 1500 on Windows and 3000 on Red Hat Linux.) This is to permit adequate space in the data space address spectrum of 2 GB for the shared class cache that is kept outside of the heap.

Figure 10-10 shows the steps to follow in RPT.

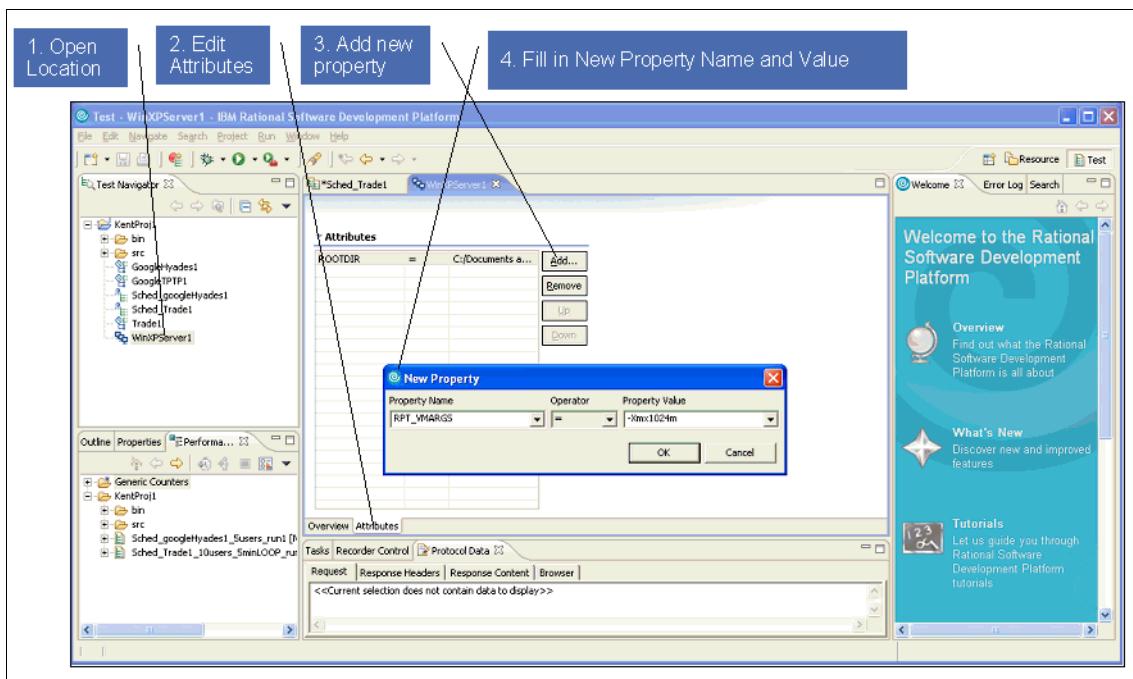


Figure 10-10 Setting the driver engine's JVM maximum heap size

10.4.2 Local driver

The steps from a local driver are really the same as a remote driver; however, when you specify in a schedule's user group that you want to run users locally (the default), RPT does not persist a location (a temporary location is created dynamically at runtime), and so there is no location asset with which to associate the RPT_VMARGS property. You can override this by creating an explicit location for running local users and specifying that location in the schedule when you want to run users locally with a JVM maximum heap value other than the default (256 MB). It is important to consider both workstation and the local driver's JVM heap sizes, and ensure that the total leave ample remaining physical memory for the operating system and any other active programs hosted on the workstation.

Figure 10-11 shows the easiest way to create a new location for local users.

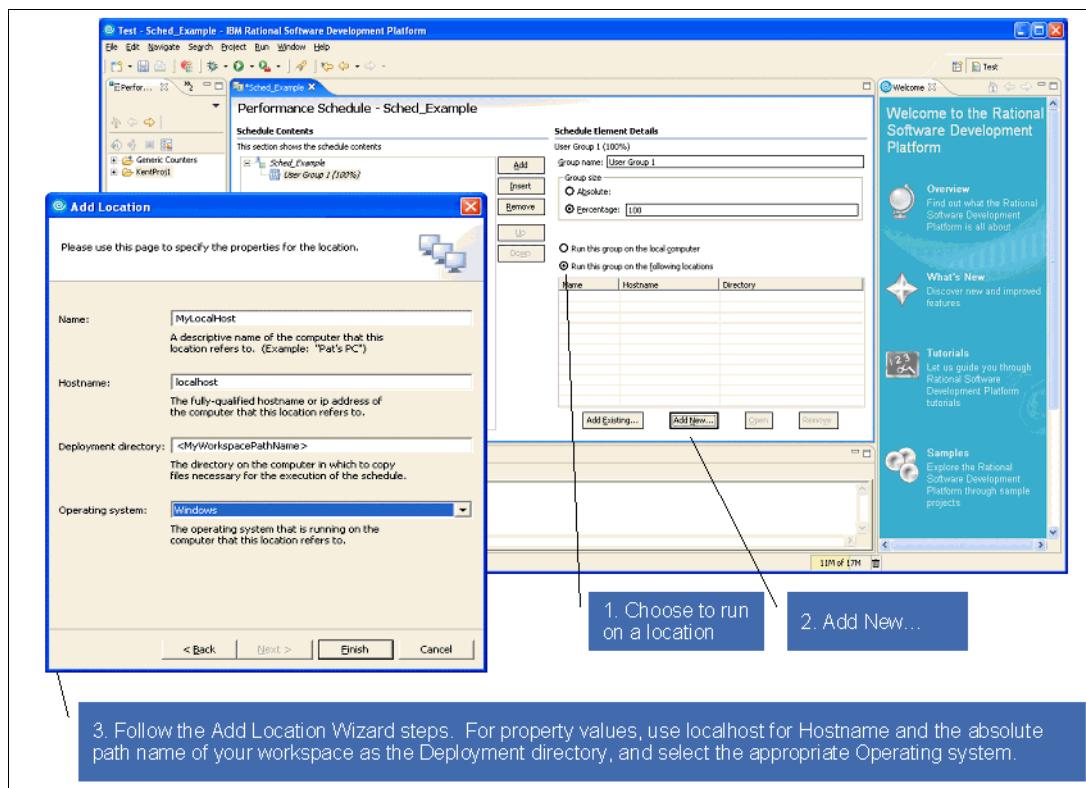


Figure 10-11 Creating an explicit location for local users

When you have created your local location, you can follow the same steps for setting the maximum heap size as shown previously for a remote driver.

10.5 RPT 6.1.2 HTTP memory footprint improvements compared to RPT 6.1.1 footprint

The 6.1.2 release of RPT offers significant HTTP scalability improvements over 6.1.1 through reductions in the driver memory footprint that are described briefly in this section, along with some comparative results. The 6.1.2 release also provided some improvements on the workbench side, which are described in the *RPT 6.1.2 Workbench Memory Optimization* document that was referenced at the beginning of this chapter (see “Rational Performance Tester 6.1.2 Workbench Memory Optimization” on page 321).

10.5.1 HTTP memory usage characteristics of RPT 6.1.1 (and 6.1)

Although previous versions of RPT were able to achieve incremental virtual user memory footprints less than 1 MB for some smaller HTTP workloads, the memory footprint did not scale well with either test length or the number of tests in a schedule. This was because earlier versions of the HTTP generated code loaded all the page element objects associated with every page into memory as the execution engine initialized each virtual user, and they remained in memory for the duration of the schedule's execution.

Although this did have some benefits in terms of greatly reducing object creation and destruction when looping in the test or schedule, the result was a very high per-user memory footprint for a variety of workloads. The longer the test recordings and the more tests added to a schedule, the higher the memory footprint. In summary, the memory footprint was approximately proportional to the number of page elements summed up over all the pages in all the tests in the schedule.

10.5.2 HTTP memory usage characteristics of RPT 6.1.2

The HTTP generated code and runtime was redesigned in 6.1.2 such that only the page elements of a single page at a time are contained in memory for each virtual user, and only for the duration of that page's response time. Objects associated with a page are created dynamically each time that page is executed. No memory is consumed by a virtual user during the time between pages, thus the HTTP runtime is now able to fully exploit the inherent scalability advantages of the hyper threaded architecture for the RPT execution engine. The redesign was done in a way that did not increase CPU overhead.

Therefore the 6.1.2 HTTP memory footprint is no longer affected by test length or the number of tests in a schedule. It is affected by the dynamics of the workload that impact the probability that a page is being executed by a virtual user, and the relationship of virtual user's activity with respect to each other, as described elsewhere in this document. But the net impact in almost all cases is a significant improvement in memory footprint over previous releases.

10.5.3 Comparison of Plants41k driver memory usage for RPT 6.1.2 versus RPT 6.1.1

In Figure 10-12, the graph of the RPT driver engine's memory footprint as a function of number of users running the Plants41k workload dramatically shows the difference we just described.

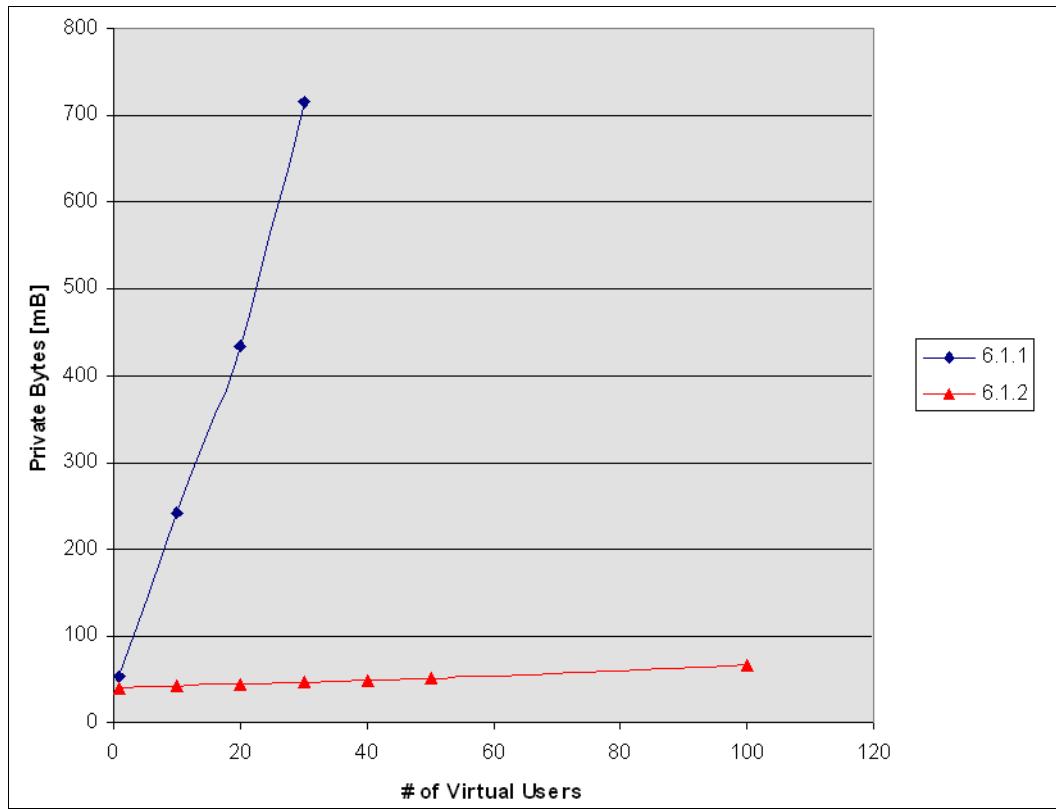


Figure 10-12 Engine memory footprint comparison (java.exe): Plants41k

10.5.4 Memory reduction of RPT 6.1.2 vs. 6.1.1 for three workloads

Table 10-17 compares the incremental virtual user memory footprint for three workloads in this document for the 6.1.1 and 6.1.2 releases.

Table 10-17 Incremental virtual user memory footprint: three workloads

| Workload | 6.1.1 [mB] | 6.1.2 [mB] | Reduction Factor (X) | % of 6.1.1 Footprint |
|----------------|------------|------------|----------------------|----------------------|
| Plants41k | 22.50 | 0.269 | 83.6 | 1.2% |
| InternetSearch | 1.40 | 0.508 | 2.8 | 36.3% |
| Atlantic | 0.51 | 0.043 | 11.8 | 8.5% |

Note that these results are for a single test in a schedule; the 6.1.2 memory reduction would be proportionally higher as the number of tests in a schedule increased.

The aforementioned reduction factors are shown in Figure 10-13.

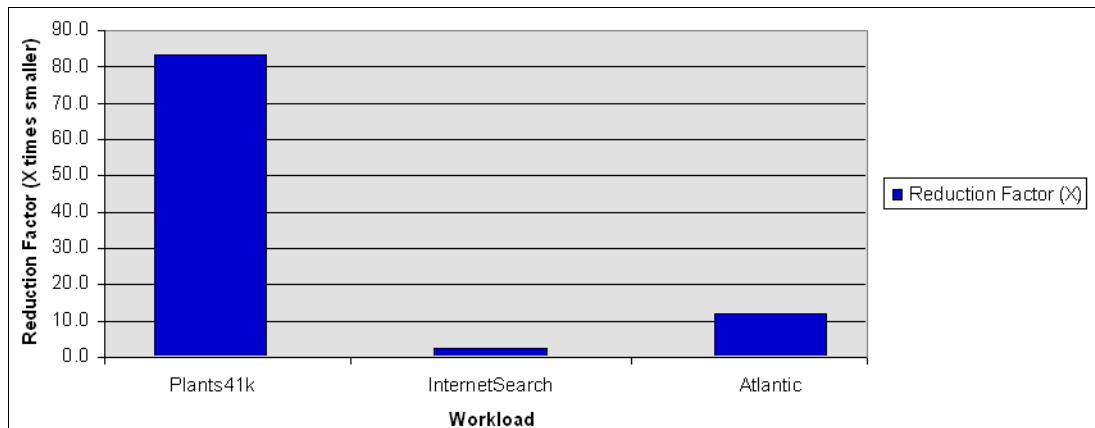


Figure 10-13 6.1.2 vs. 6.1.1 incremental virtual user memory footprint reduction factor

10.6 Intermediate results for memory footprint measurements

Showing the results for each step along the way for calculating the incremental virtual user memory footprint for each of the workloads and variations in 10.2.3, “Memory footprint measurements” on page 339 would be excessive for the purposes of this chapter. However, to illustrate the methodology described in 10.1.7, “Methodology for determining the incremental virtual user memory footprint” on page 329, intermediate results are shown for two of the workloads, TradeSVTApp and InternetShoppingCart.

10.6.1 TradeSVTApp

In this section we show the results for the TradeSVTApp workload.

Perfmon data for individual runs

The perfmon data from each run is shown in the following sections.

For this workload, the individual collecting the data in a couple of cases used two separate perfmon sessions using 5 and 30 second sample intervals respectively instead of the 5 and 60 second sample intervals mentioned in the methodology. The 30 second sample is still fine for our purposes, and is probably less than a half-percent higher than what would have been obtained using a 60 second sample interval.

The Private Bytes counter is selected in each figure, so that its Last/Average/Minimum/Maximum counter values can be read directly; in the graph the selected Private Bytes line is highlighted in white.

The methodology that is followed is to select the maximum value of Private Bytes for the established steady-state duration of the run (at least several test iterations after all users are active), prior to the presence of any unnatural elevation, if any. In all cases below this equates to the maximum value of Private Bytes for the entire run, which can be simply read directly from the Maximum counter value.

In Figure 10-14, the selected value of Private Bytes is 43556864, rounded to 43.6 kB.

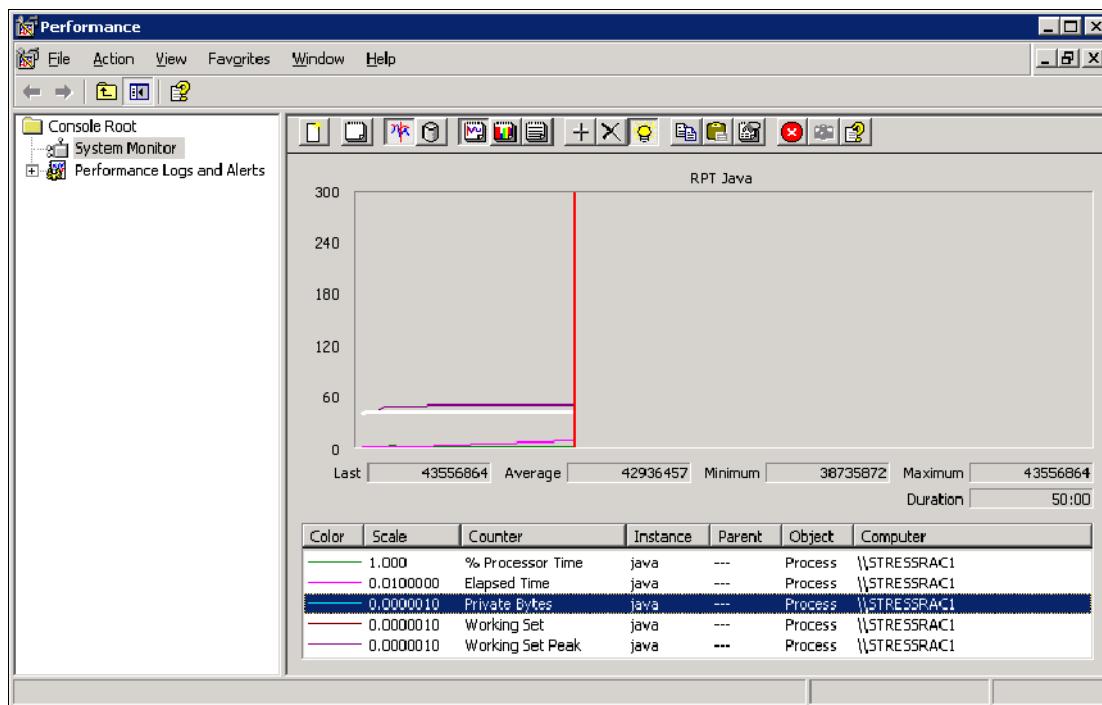


Figure 10-14 One virtual user (entire run with 30-second sample interval)

Figure 10-15 shows another straightforward case; the selected value of Private Bytes is 46432256, rounded to 46.4 mB.

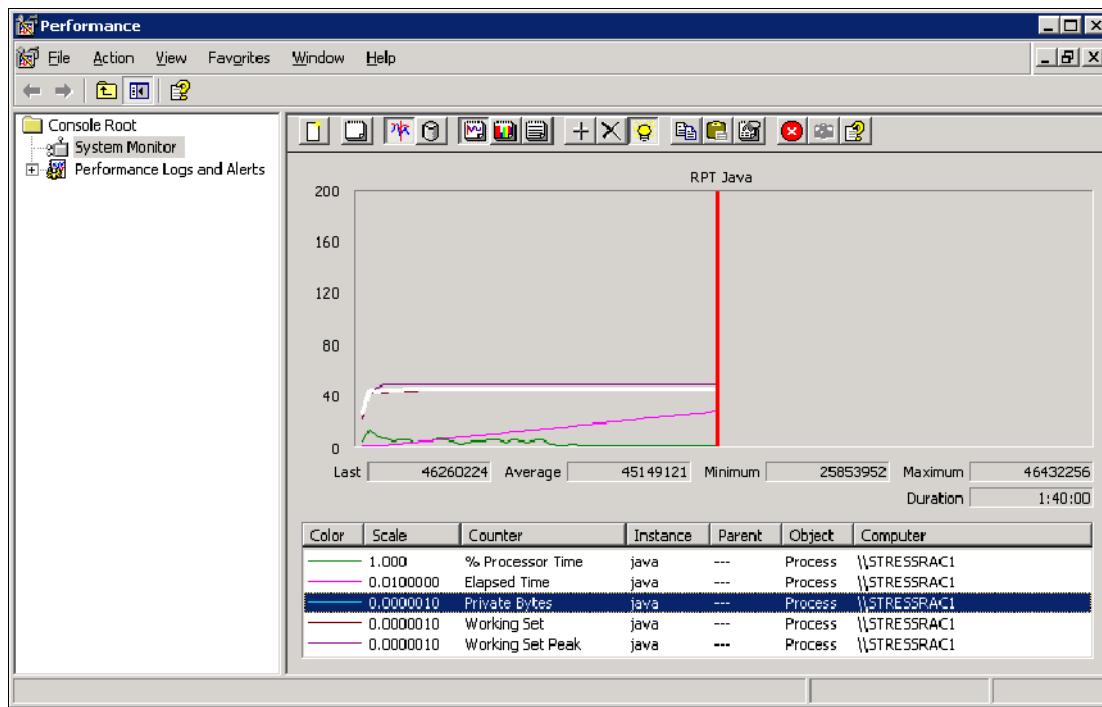


Figure 10-15 10 virtual users (entire run with 60-second sample interval)

In Figure 10-16, the selected value of Private Bytes is 62345216, rounded to 62.3 mB.

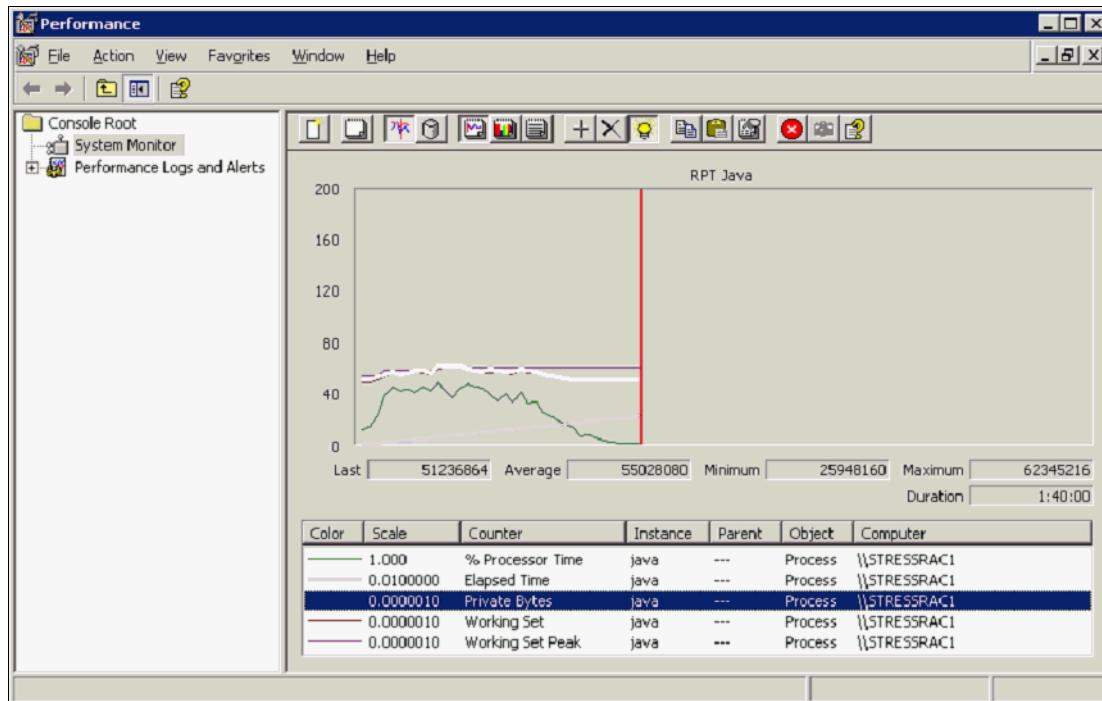


Figure 10-16 100 virtual users (entire run with 60-second sample interval)

In Figure 10-17, the selected value of Private Bytes is 77774848, rounded to 77.8 mB.

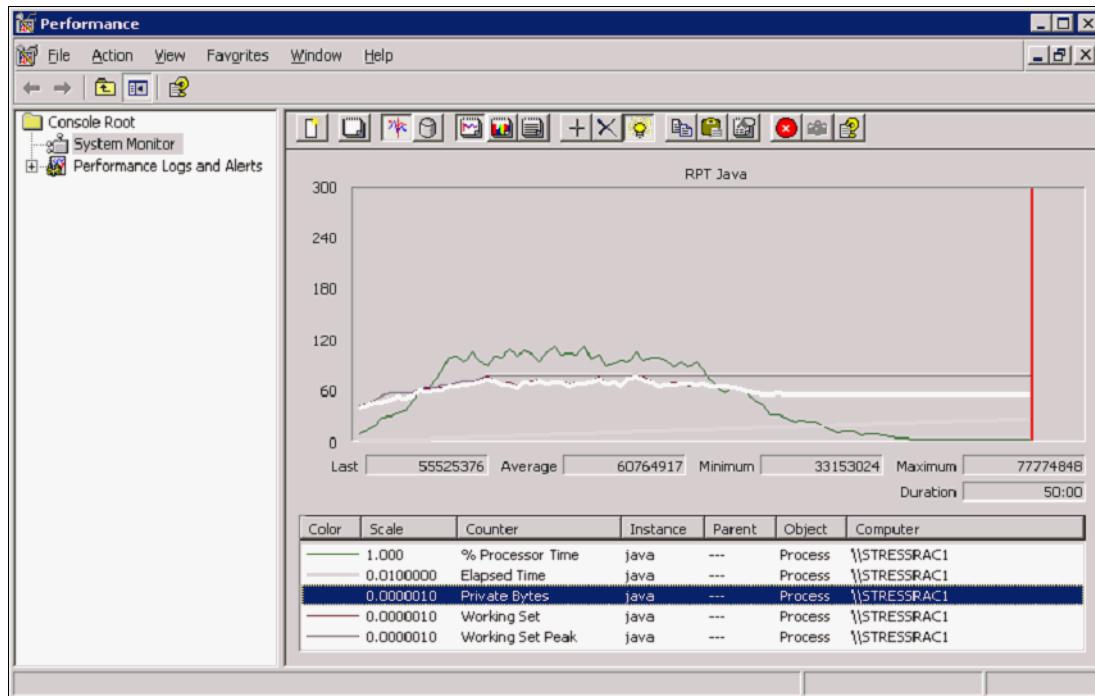


Figure 10-17 200 Virtual Users (Entire run with 30 second sample interval)

Calculating the incremental virtual user memory footprint from the individual runs

The data from the previous individual runs was placed in an Excel spreadsheet and the intercept and slope calculated using a best-fit linear regression line from the actual known data points (using INTERCEPT and SLOPE formulas of Excel). Using the slope and intercept, the best-fit y-values (Private Bytes) are calculated for the actual x-values (# of Virtual Users) in an adjoining column labeled 6.1.2 Fitted. A percentage difference—(actual-fitted)/fitted—is calculated to help determine the acceptance of outliers, looking for agreement within a few percentage points. All of this data is shown in Table 10-18.

Table 10-18 Incremental virtual user memory footprint from individual runs

| # Virtual Users | Private Bytes [mB] | | % Delta |
|-----------------|--------------------|--------------|---------|
| | 6.1.2 Actual | 6.1.2 Fitted | |
| 1 | 43.6 | 44.5 | -1.97% |
| 10 | 46.4 | 46.0 | 0.86% |
| 100 | 62.3 | 61.3 | 1.62% |
| 200 | 77.8 | 78.3 | -0.65% |

| | | |
|-----------|-------|-------|
| Intercept | 44.31 | 44.31 |
| Slope | 0.170 | 0.170 |

The slope of 0.170 is the reported value for the incremental virtual user memory footprint for the TradeSVTApp workload in the “Memory footprint for base workloads” on page 339.

The actual and fitted values are shown in Figure 10-18:

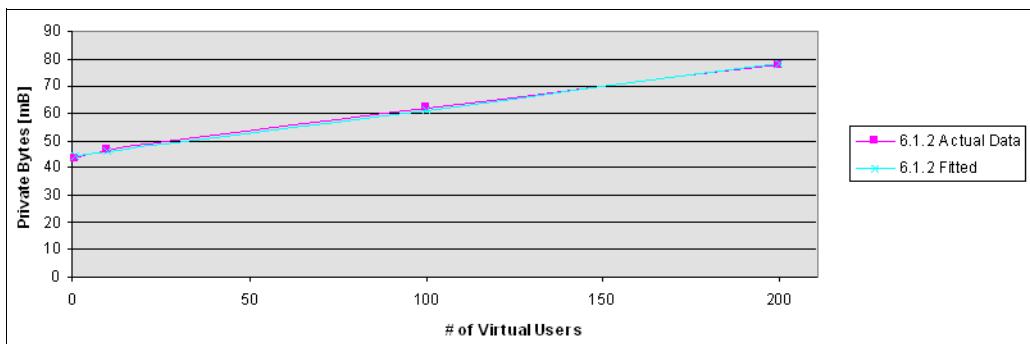


Figure 10-18 Engine memory footprint (java.exe): TradeSVTApp

Figure 10-18 shows a desirable linear relationship in the memory footprint over the range of virtual users measured. The intercept value of 44.3 is high compared to other workloads. This is explained by the fact that this is the only workload that includes datapools, and in fact there are 4 of them, ranging from 100 to 500 rows in size. The datapool data is loaded once and shared across all virtual users and therefore is overhead independent of the number of users, which should in fact show up in the intercept value, which it does.

10.6.2 InternetShoppingCart

In this section we show the results for the InternetShoppingCart workload.

Perfmon data for individual runs

The perfmon data from each run is shown in the subsections that follow. An explanation of the process was provided previously in the corresponding section for the TradeSVTApp workload (“Perfmon data for individual runs” on page 369). A difference in the perfmon graphs from that section is that the highlight color for the graphs in this section is black instead of white.

Note that there is no *one virtual user run* for this workload, which generally is not all that important. (One was actually made to validate the test was running correctly, but the schedule settings did not match the other runs, so it was not helpful to include it.)

In Figure 10-19, the selected value of Private Bytes is 47378432, rounded to 47.4 mB.

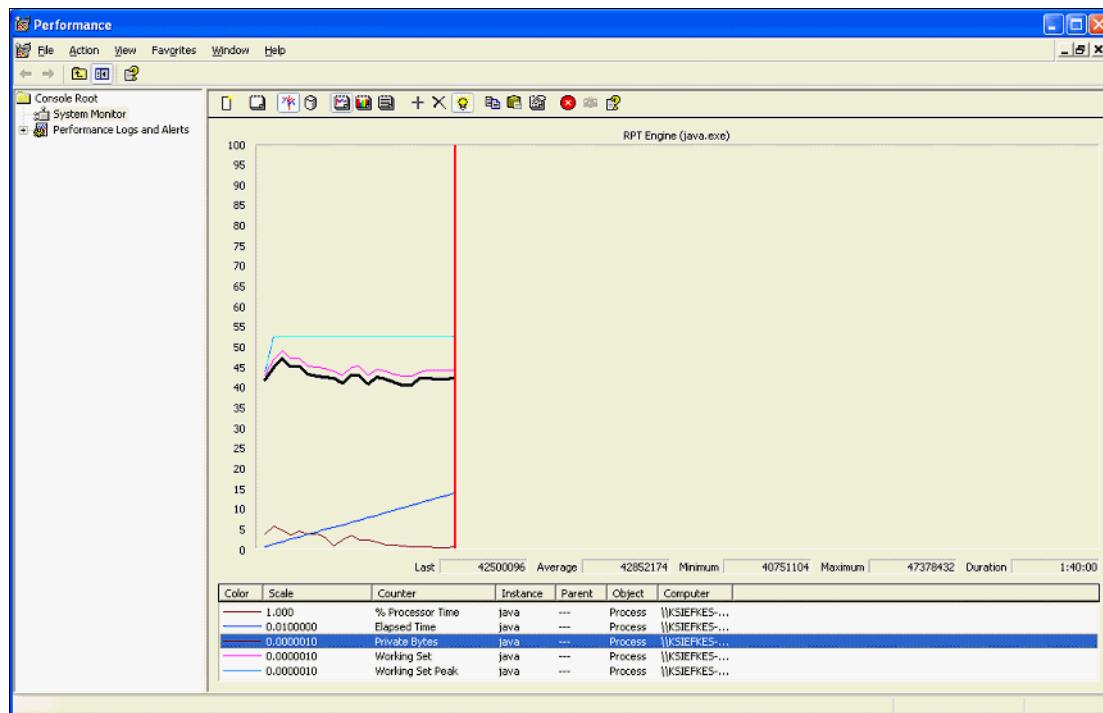


Figure 10-19 10 virtual users (entire run with 60-second sample interval)

In Figure 10-20, the selected value of Private Bytes is 49963008, rounded to 50.0 mB.

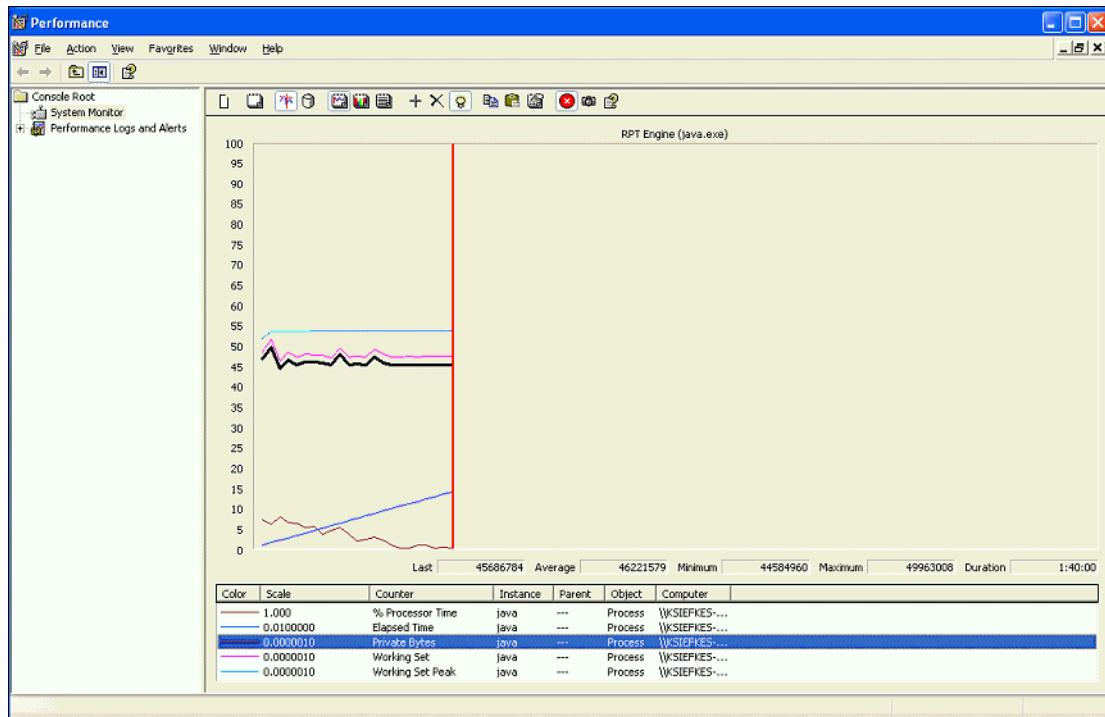


Figure 10-20 20 virtual users (entire run with 60 second sample interval)

In Figure 10-21, the selected value of Private Bytes is 58114048, rounded to 58.1 MB.

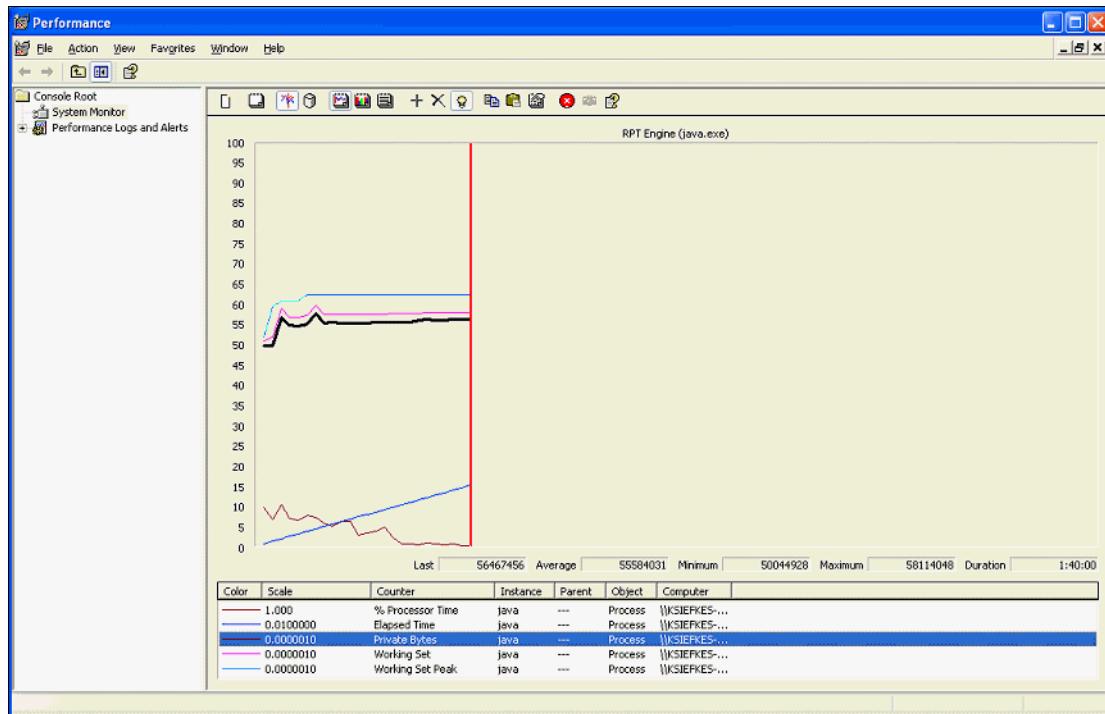


Figure 10-21 30 virtual users (entire run with 60-second sample interval)

In Figure 10-22, the selected value of Private Bytes is 65564672, rounded to 65.6 mB.

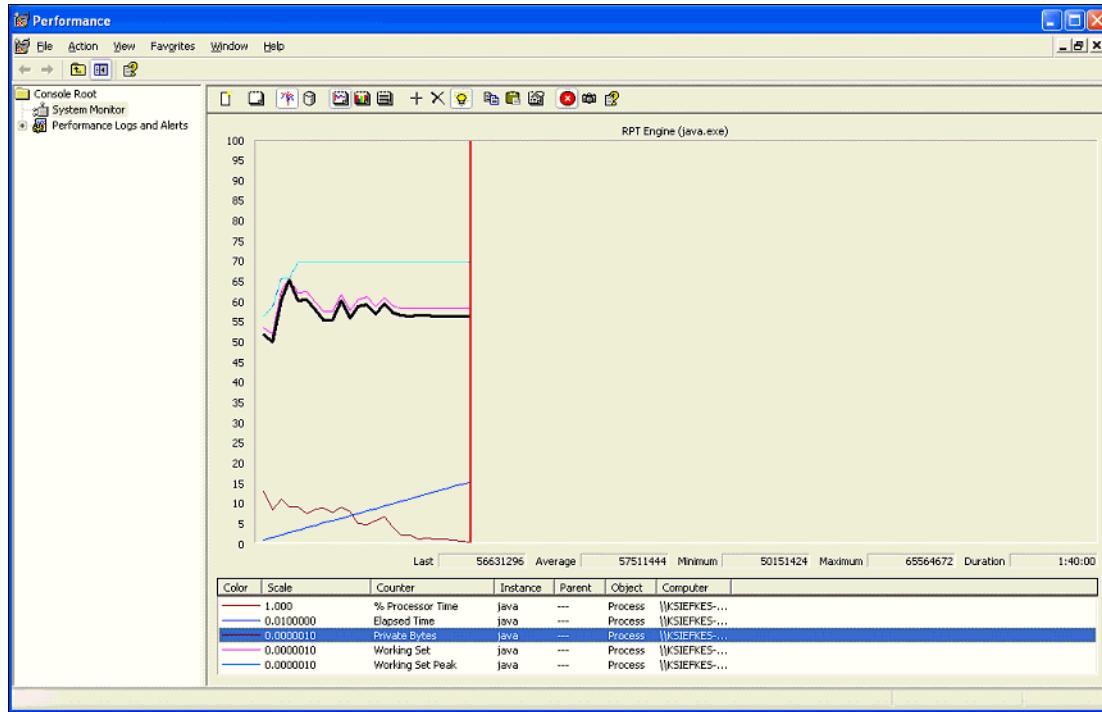


Figure 10-22 40 virtual users (entire run with 60-second sample interval)

Calculating the incremental virtual user memory footprint from the individual runs

Table 10-19 shows the first calculation produced, following the process used and described previously in the corresponding section for the TradeSVTApp workload ("Calculating the incremental virtual user memory footprint from the individual runs" on page 373).

Table 10-19 Initial calculation

| # Virtual Users | Private Bytes [mB] | | % Delta |
|-----------------|--------------------|--------------|---------|
| | 6.1.2 Actual | 6.1.2 Fitted | |
| 10 | 47.4 | 45.9 | 3.34% |
| 20 | 50.0 | 52.1 | -4.10% |
| 30 | 58.1 | 58.4 | -0.53% |
| 40 | 65.6 | 64.7 | 1.42% |

| | | |
|-----------|-------|-------|
| Intercept | 39.60 | 39.60 |
| Slope | 0.627 | 0.627 |

The relatively large percentage difference in the % Delta column for the first two points indicates a potential discrepancy that needs a closer look. When plotted in the graph (Figure 10-23), it can be seen clearly that the 10 user value is bringing the left side of the fitted curve up and lowering the slope.

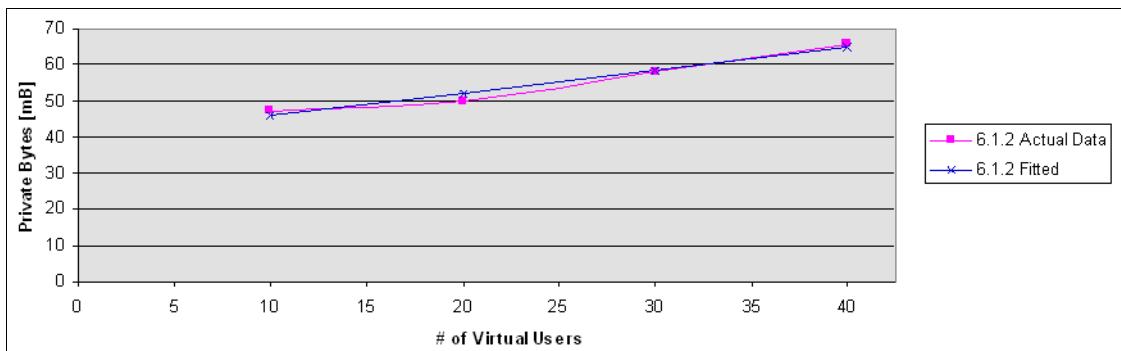


Figure 10-23 Engine memory footprint (java.exe): InternetShoppingCart (first attempt--discarded)

By discarding this value from the calculation of the intercept and slope a very good fit is obtained using the other three data points, and the 10-user fitted value is 42.3 (12% less than the discarded value of 47.4). The slope increased significantly after this change, up from 0.627 to 0.780.

Although data should not be discarded casually, one has to remember that the purpose of the exercise is prediction and any choices that will improve the confidence associated with that prediction are generally good ones. The following reasons validate this choice:

- ▶ This data point was the clear outlier, and discarding it produced an excellent fit, which all remaining deltas less than 0.4%.
- ▶ If in doubt, give higher importance to the measurements at higher user counts and less importance to the lower user count measurements. There is typically more *noise* in the lower user count measurements especially when using random delays such as the paced loop control for this workload, and more importantly, the higher user counts should carry more weight in a sizing goal of predicting the memory usage at much higher user counts.
- ▶ It is probably better to err on the safe side when establishing HW needs (although that might depend upon the availability and cost of acquiring the HW), and in this case the calculated slope after discarding the 10-user value is a full 20% higher.
- ▶ The intercept value of 35.3 for this workload (no datapools) is closer to the expected value than 39.6.

Note that although it was not possible in this case to go back and rerun the test with similar conditions, that would be the first choice, to see if the outlier went away or was consistently present.

The final adjusted data is shown in Table 10-20.

Table 10-20 Final calculation

| # Virtual Users | Private Bytes [mB] | | % Delta |
|-----------------|--------------------|--------------|---------|
| | 6.1.2 Actual | 6.1.2 Fitted | |
| 1 | n/a | 35.3 | |
| 10 | discarded | 42.3 | |
| 20 | 50.0 | 50.1 | -0.20% |
| 30 | 58.1 | 57.9 | 0.35% |
| 40 | 65.6 | 65.7 | -0.15% |

| | | |
|-----------|-------|-------|
| Intercept | 34.50 | 34.50 |
| Slope | 0.780 | 0.780 |

The slope of 0.780 is the reported value for the incremental virtual user memory footprint for the InternetShoppingCart workload in “Memory footprint for base workloads” on page 339.

The actual and fitted values are plotted in Figure 10-24 (labeled Final), which shows a desirable linear relationship in the memory footprint over the range of virtual users measured.

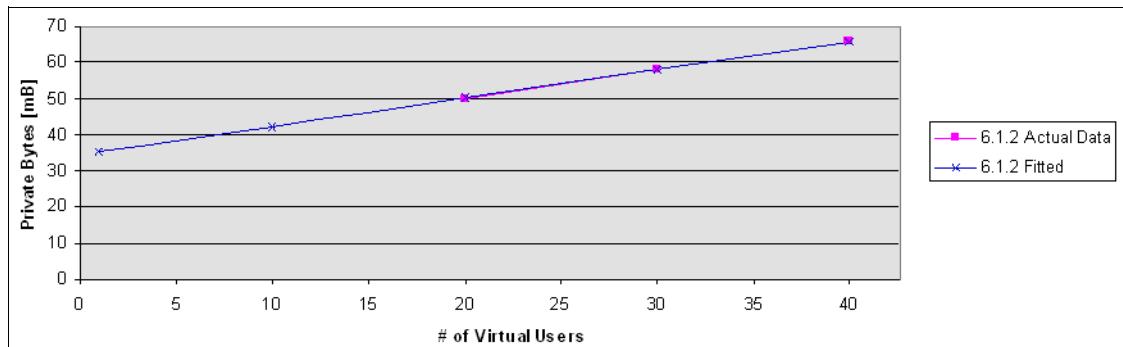


Figure 10-24 Engine memory footprint (java.exe): InternetShoppingCart (final)



Testing SAP enterprise applications

This chapter describes how IBM Rational Performance Tester can be used to test SAP enterprise applications. We discuss how to test the SAP GUI for Windows, which is the most popular SAP front end.

We highly recommend that you have a good understanding of Chapters 1 through 7, Chapter 11, and Chapter 12 before you begin testing SAP enterprise applications.

We cover the following topics:

- ▶ SAP performance test prerequisites
- ▶ Recording, editing, and execution of SAP tests
- ▶ Building complex tests
- ▶ Tips and tricks

11.1 Introduction

In this chapter we describe how RPT can be used to test SAP enterprise applications. SAP provides a comprehensive range of enterprise applications to empower every aspect of customer business operations. SAP enterprise applications are listed in Figure 11-1.



Figure 11-1 SAP enterprise applications

An overview of SAP applications can be found at:

<http://www.sap.com/solutions>

Although we can find more than 50 applications there, the number of front ends is limited. The foundation of these solutions is the SAP NetWeaver® platform. The architecture of SAP NetWeaver Application Server is shown in Figure 11-2.

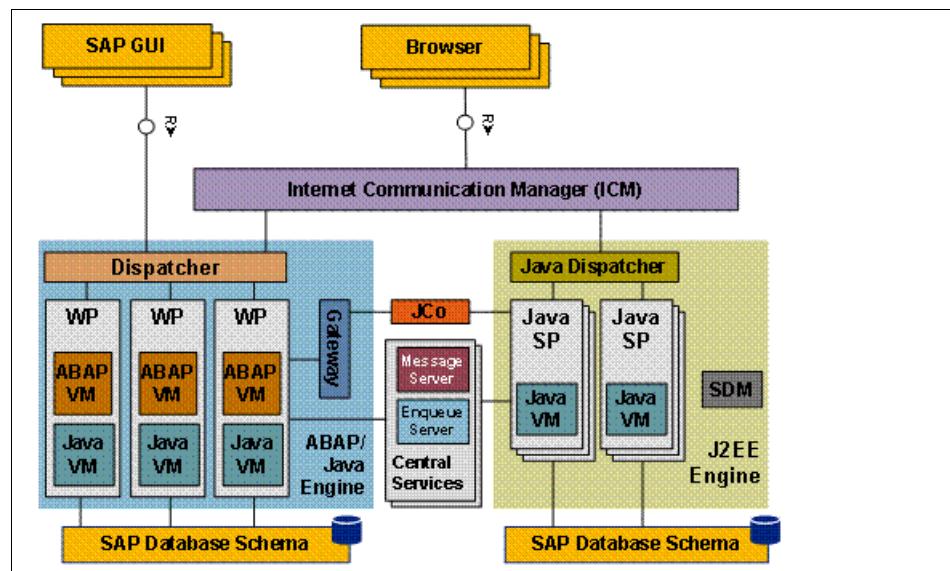


Figure 11-2 Architecture of the SAP NetWeaver Application Server

From a technical perspective, we can access the SAP system using the SAP GUI or a standard Web browser:

- ▶ SAP GUI for Windows:

This is the most popular SAP GUI. Nearly every customer is using this front end. It is the fastest SAP GUI and has to be installed on the local computer. In this chapter we concentrate on testing the SAP GUI for Windows.

- ▶ SAP GUI for HTML or Browser:

This is the browser front end. It does not require any additional SAP installations on the local computer. This front end can also be tested as it uses the HTTP protocol.

- ▶ SAP GUI for Java:

This version does not depend on the local computers operating system. This is the less frequently used front end.

Note: In this chapter we use a SAP International Demonstration and Education System (IDES) for all recordings and playbacks. Having such a system available allows you to reproduce all examples.

11.2 Testing the SAP GUI for Windows

In this section we explain performance tests of the SAP GUI for Windows. The topics covered in this section include:

- ▶ Prerequisites
- ▶ Recording, editing, and execution of SAP tests
- ▶ Best practices
- ▶ Troubleshooting

11.2.1 SAP GUI for Windows prerequisites

Before you can start recording or executing performance tests, you must check a couple of prerequisites. The SAP GUI for Windows software must be installed on the same computer as RPT. The SAP GUI for Windows is required for recording and execution of performance tests.

If you want to simulate a large number of users, you have to deploy tests on remote computers. The following software must be installed on each remote computer:

- ▶ The SAP GUI for Windows software, configured with the same logon properties as the client on which the tests were recorded
- ▶ The Agent Controller that is provided with RPT

Note: You cannot use Linux or AIX computers for recording or execution of SAP performance tests. Only Windows computers are supported.

Supported environments

The following versions of the SAP GUI for Windows are supported:

- ▶ SAP GUI for Windows 6.20 with patch level 44 or higher
- ▶ SAP GUI for Windows 6.40 with patch level 13 or higher

You can verify the version as follows:

- ▶ Start the SAP Logon Panel (Figure 11-3): *Start → Programs → SAP Front End → SAP Logon.*



Figure 11-3 SAP Logon Panel

- ▶ Click on the left icon in the title bar, which opens the menu. Select *About SAP Logon*.

- ▶ Now verify the version information as shown in Figure 11-4. It is version 6.40 with patch level 17. This is a supported environment.



Figure 11-4 SAP Version Information

Note: You can also verify the version information directly in the SAP GUI if you are already logged on. Enter *ALT+F12* and select *About* from the menu.

Enable scripting

Performance test recording and execution requires scripting to be enabled on the SAP server and on all SAP GUI clients that are installed on remote computers. The following instructions are for SAP version 6.40 and might vary with other versions. Refer to SAP documentation for further information.

Performance testing relies on the SAP Scripting API and ActiveX®. Make sure that these options are selected when installing the SAP GUI client.

To configure SAP for performance testing, scripting must be enabled on the server and on the client:

- First we enable scripting on the server:
 - Run the SAP GUI client and logon to SAP with your user ID and password. Administrator privileges might be required to enable scripting on the server.
 - In SAP, run the transaction `rz11`, type the parameter name `sapgui/user_scripting`, and then click *Display*. If the parameter is not found, then make sure you have the correct support package level from SAP. Contact your SAP representative for guidance.
 - Now verify the *Current value* (Figure 11-5). If the value is FALSE, click the *Change value* button, and then set the *New value* to TRUE in uppercase characters.

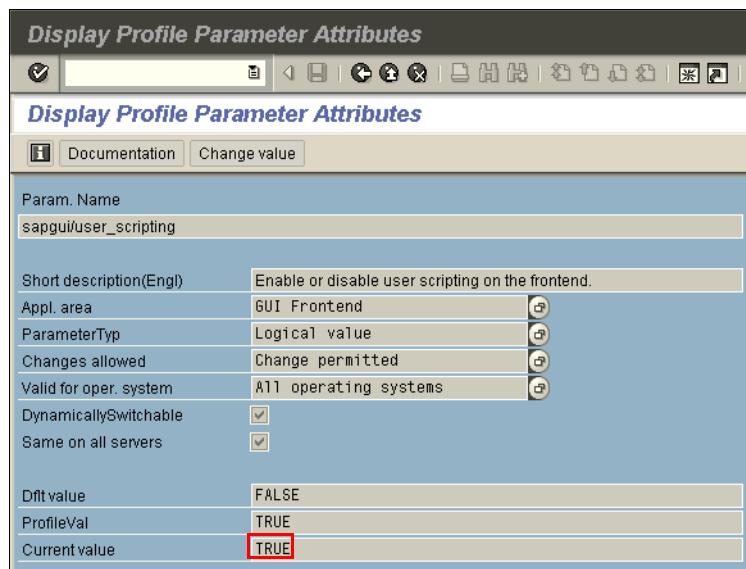


Figure 11-5 SAP GUI user scripting parameter

- Click *Save*, and then end the transaction. Scripting will be enabled the next time you log on.

- ▶ Next we also must enable scripting on the client:
 - In the SAP GUI client toolbar, click the *Customizing of Local Layout* toolbar icon or *ALT+F12*, and then select *Options*.
 - Select the *Scripting* page.
 - Select *Enable Scripting*, and then clear both *Notify When a Script Attaches to Running GUI* and *Notify When a Script Opens a Connection* (Figure 11-6).

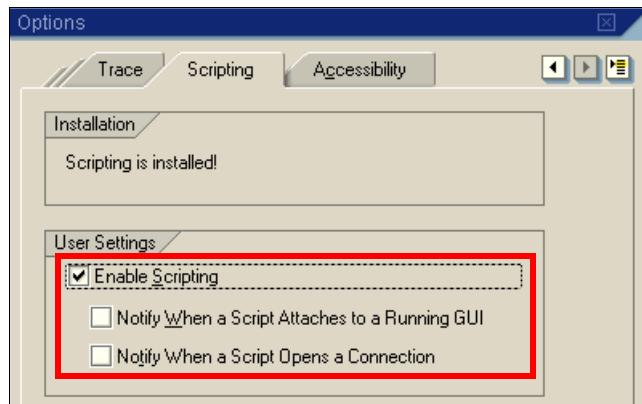


Figure 11-6 SAP GUI front end scripting settings

- Click *OK*.
- In the Help menu, select *Settings*, and then select the *F4 Help* page.
- In *Display*, select *Dialog (modal)* and then click the button.

11.2.2 Recording SAP GUI tests

We are now ready to record a SAP performance test. We can record a SAP test directly from the SAP GUI. When we record, the recording wizard automatically starts the SAP GUI and records all the interactions that occur between the GUI and the server.

Note: Ensure that the session that you are recording will be reproducible. For example, if you create items in SAP and do not delete them, then they will already exist when the test is run, which might cause the test to fail.

Record a new SAP performance test

RPT can record a new or into an existing performance test. We start recording a new performance test:

- ▶ In the Test perspective, select *File → New → Test from Recording*, or click the  button.
- ▶ Select *Create test from new recording* and *SAP Recording*. Click *Next* (Figure 11-7).

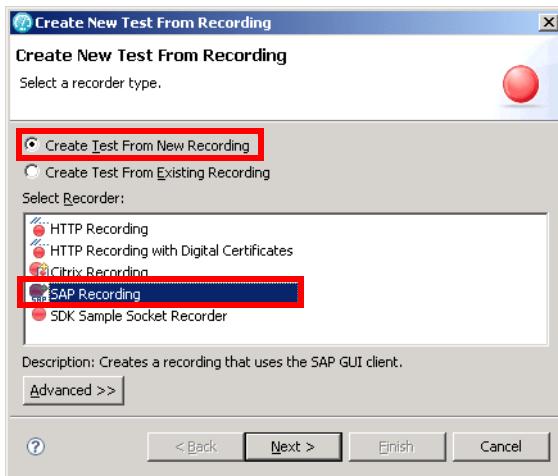


Figure 11-7 Create new SAP Test

- ▶ For Recording file name, type a name for the test. In our example, we record the FS10N transaction, so we type FS10N, and click *Next*.
- ▶ Select how you want to connect to the SAP server:
 - In most cases, select *SAP Logon*. In *SAP system name*, enter the description used by SAP Logon to identify the server (Figure 11-8).
 - If your environment does not support SAP Logon, select *Connection by string*. For Application server, enter the host name or IP address of the server, and specify the System Number and other options if required. Refer to the SAP documentation for details about the other SAP Logon options.
 - If the environment uses gateways or routers to connect to the SAP R/3 server, select *Connection by string*. For Application server, enter the full connection string, for example:
`/H/gate.sap.com/S/3299/H/172.16.63.17/S/3200`
 - Leave System Number and other options blank.

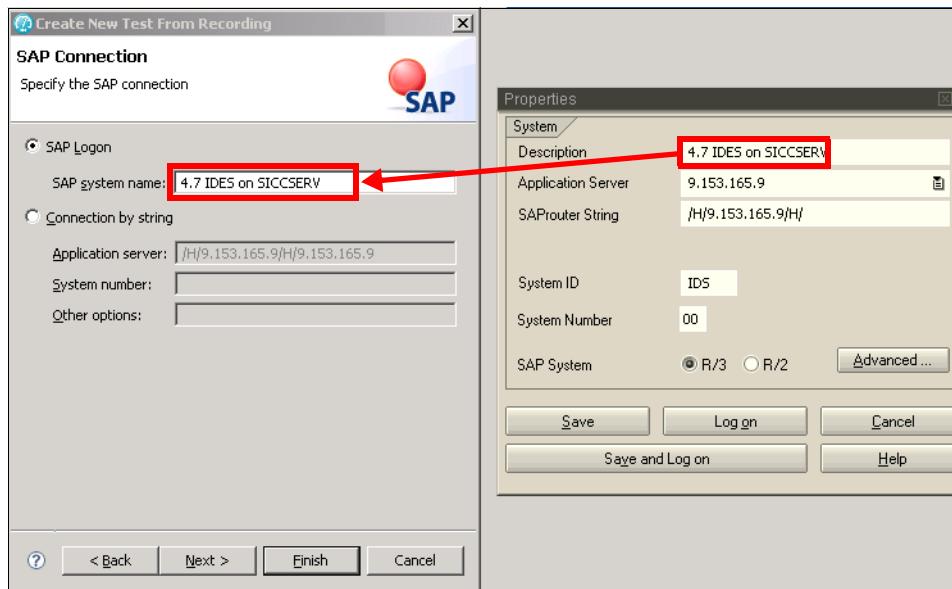


Figure 11-8 Specify the SAP connection

- If this is the first time you record a SAP performance test, read the Privacy Warning and select *I accept* to proceed (Figure 11-9).



Figure 11-9 Privacy Warning

- ▶ Click *Finish* to start recording. A progress window opens while the SAP GUI starts. In some cases, you might see a warning that a script is opening a connection to SAP. To avoid this situation, follow the description in “Enable scripting” on page 387.
 - ▶ Log on to SAP and perform the transactions that you want to test.
- We are recording an example with the FS10N transaction:
- Log on to SAP using a valid user ID/password combination. Also type in the language.
 - Type FS10N and click or press Enter.
 - In the next step, we are using the following values: G/L account 3100000, Company code 3000, Fiscal year 2001. Click *Execute* or press F8.
 - Click *Cancel* or press F12.
 - Close the SAP GUI or select *System → Log off*.
 - In the Log Off dialog, click *Yes*.
 - ▶ When you have completed the transactions to be tested, stop the recorder. You can do this by closing the SAP GUI or by clicking *Stop* in the Recorder Control view.
 - ▶ For security reasons, the password cannot be recorded by the SAP test recorder. Instead, it is requested at the end of the recording session. In the Enter Password dialog, enter the password for the account used for recording (Figure 11-10). This is required because the SAP GUI does not allow direct recording of the password. A progress window opens while the test is generated. On completion, the Recorder Control view displays the message Test generation completed, the Test Navigator lists the test, and the test opens in the test editor.

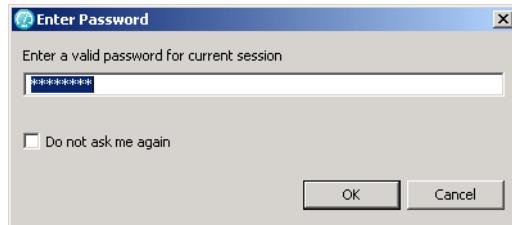


Figure 11-10 RPT Enter Password dialog

Note: It is required that you provide a password for the SAP performance test and you should do this here. Otherwise you have to do this manually in the test editor.

The generated SAP performance test of our example is shown in Figure 11-11.

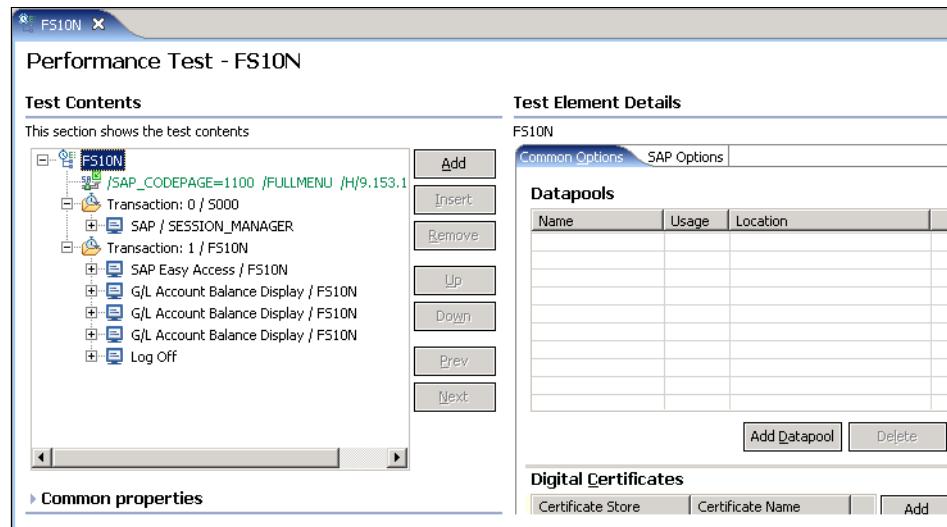


Figure 11-11 SAP performance test editor

If you have provided a password, then this performance test is ready for playback. No additional changes are necessary.

Record into an existing SAP performance test

It is also possible to record into an existing SAP performance test. Use this feature to add or replace a part of a recorded session.

Inserting a new sequence into a test requires that the SAP session reaches the same state as is expected at the point where the new sequence is inserted. To do this, the SAP test recorder automatically replays the existing scenario up to the insertion point before starting the new recording.

To insert a recording into a test, perform the following steps:

- ▶ In the test editor, select the element before which you want to insert the new recording. It is easier to manage the new test sequence when the insertion point is at the transaction level of the test.
- ▶ Click *Insert*, and then *New recording*. The test starts replaying up to the selected insertion point.
- ▶ When the New Recording window is displayed, perform the sequence of actions that you want to add to the existing test (Figure 11-12).

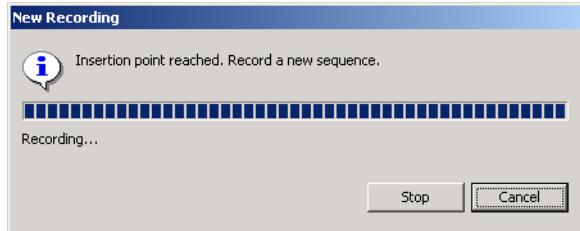


Figure 11-12 New SAP recording

- ▶ When you have finished, in the New Recording window, click *Stop* to stop the recording. A progress window opens while the test is generated. On completion, the Recorder Control view displays the message, *Test generation completed*, and the test is updated with the new content. You can also click *Cancel* to cancel the recording.
- ▶ After the test has been updated in the Test Navigator, verify that the new sequence was properly inserted into the test, and then select *File* → *Save* to save the test or select *File* → *Revert* to cancel the inserted recording.

11.2.3 Editing SAP GUI tests

A detailed description of editing SAP GUI tests can be found in 4.3.3, “Editing SAP tests” on page 58. You should have read this before continuing here. The main difference is that in HTTP we have pages and page elements in the script, whereas in SAP tests we have the concept of screens.

Prepare a multi-user playback

Although the test we recorded in the previous chapter is ready for playback, it cannot be used for a multi-user playback at the moment. As do most enterprise applications, SAP requires different unique user IDs.

If you create a performance schedule and start it with more than one user, only the first user can logon. The other users will get a warning on multiple logons (Figure 11-13). Users have now the choice to continue the logon without terminating the existing sessions.

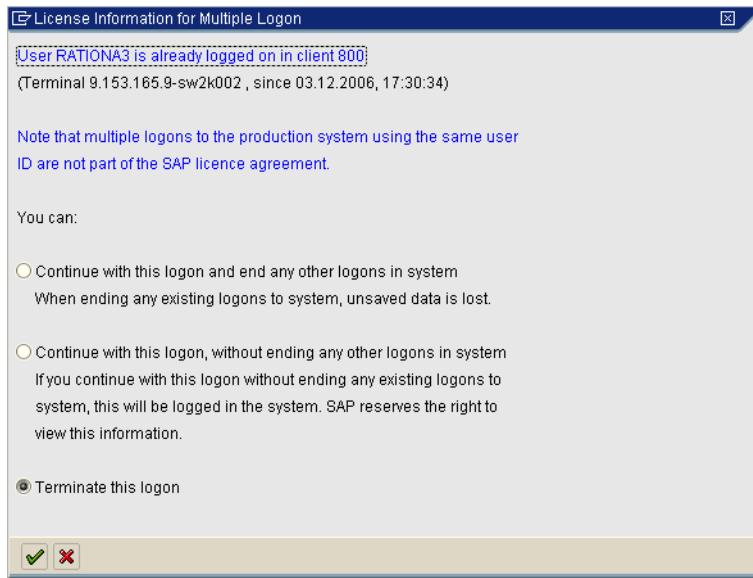


Figure 11-13 SAP multiple logon warning

We have the following choices to avoid the multiple logon message:

- ▶ All virtual testers must use unique user IDs. In this case we have to create at least the number of SAP logons as we plan to use in the schedule with the highest number of users. This can be a very time consuming activity if we do this manually. But we can use RPT or Rational Functional Tester (RFT) to do this for us. We can record a SAP test in RPT as follows:
 - Logon to SAP and start the su01 transaction to create new users.
 - Into the user field type in the name of an existing user. Now click *Copy* (or press Shift+F5). It is easier to copy users than to create new ones.
 - In the dialog window type in the to field the new logon names and press *Enter*.
 - Type in an initial password into the required fields and click *Save* (Ctrl+S).

The script now must be enabled to use a datapool with logon names. Then we have to create a schedule to create our SAP logons. After the schedule is finished, the logons are almost ready. We have to logon once to change the initial password, which is required by SAP. This can also be automated using RPT.

- ▶ Suppress the multiple logon message by setting an additional parameter in the instance profile. Unfortunately this cannot be done using the SAP GUI. The system administrator must add the parameter `login/multi_login_users` to the default system profile `default.pfl` directly. In this parameter, enter the names of all the users separated by "," (comma) for which you want to suppress the multiple logons message. You must enter the names in uppercase as in this example:

```
login/multi_login_users=USER1,USER2
```

11.2.4 Executing SAP GUI tests

This is almost the same as executing HTTP tests, which is covered in Chapter 5, “Test playback and debugging” on page 139. There are some exceptions. In the test editor there is an option that has an influence on the execution, and the reports used for a SAP playback are a bit different. The differences in the reports are covered in “Evaluating SAP GUI test runs” on page 397.

Display SAP GUI on execution

In the test editor we can determine if the SAP GUI is displayed at execution time (Figure 11-14).

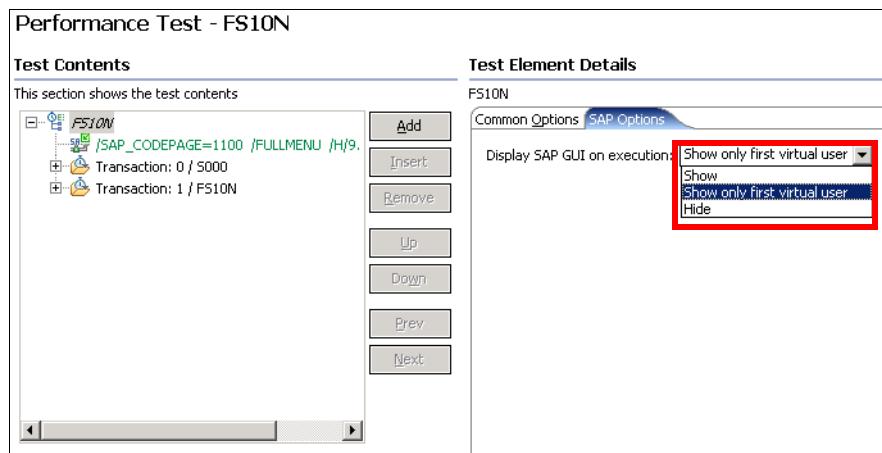


Figure 11-14 Display SAP GUI at execution time

During test execution, it might not be desirable to display the SAP GUI. Hiding the SAP GUI improves the performance of the virtual users. This setting specifies the behavior for the current test suite. However, you can change the default setting for generated tests in the SAP Test Generation preferences.

You can select one of these options:

- ▶ *Hide*: When selected, all instances of the SAP GUI are hidden. In some cases, modal dialog boxes from the SAP GUI can flash briefly on the screen. This is the default setting.
- ▶ *Show*: When selected, the SAP GUI is displayed for all virtual users.
- ▶ *Show only first virtual user*: When selected, the SAP GUI is displayed only for the first virtual user. All other instances of the SAP GUI are hidden. This allows you to monitor the execution.

Normally you should select *Hide*, because this setting guarantees the best performance during test execution. Otherwise, if you are still developing SAP tests, it is a good choice to select *Show only first virtual user*.

Note: Make sure that you set *Display SAP GUI on execution* to *Hide* in the SAP Options for every SAP test in your performance schedule.

11.2.5 Evaluating SAP GUI test runs

Results analysis and reporting was already discussed in Chapter 7, “Results analysis and reporting” on page 181. The difference is that during a playback of a SAP GUI test, RPT automatically displays the SAP Performance Report. If we want to analyze the verification points, we must use the SAP Verification Point Report.

11.2.6 Creating complex SAP GUI tests

In this section we discuss a bit more complex SAP GUI tests. Most performance tests should model typical user scenarios. A typical SAP user only performs a logon once a day. The user is doing a lot of transactions the whole day and at the end of the day the user closes the SAP GUI.

With RPT it is possible to record several transactions into a SAP test. But the logon and the logout are limited to a SAP performance test. So if you create a performance schedule and include some SAP performance tests, you will simulate a lot of logons. Every SAP performance test is creating a new session.

Obviously this is not what a realistic user is doing. We have to think about a possibility to allow a virtual user creating a session only once at the beginning of the test. Because a session is limited to a SAP performance test, we have to create a complex SAP performance test, as shown in Figure 11-15.

The screenshot shows the SAP Performance Test interface for a complex test named 'Complex1'. The left pane displays a hierarchical tree of test elements, including a transaction 'Login', a loop with 30 iterations containing transactions 'F510N', 'K503', and 'FI03', and a logout transaction. The right pane, titled 'Test Element Details' for a 'Conditional(IF) Block', shows configuration for an 'If' statement. It includes fields for 'First operand' (set to 'Custom code: custom.RandomSelector'), 'Selected data type' (set to 'Custom Code'), 'Operator' (set to 'Equals'), 'Second operand' (set to '1'), and 'Selected data type' (set to 'Text value'). Below these are 'Options' for negation and case sensitivity.

Figure 11-15 Complex SAP performance test

We can create this performance test by recording the three transactions into one test and then inserting the additional elements, such as the loop and the if statements. Another option is using copy/paste to get additional transactions into a performance test.

The first element of the complex test is the logon. After this we have a loop with 30 iterations followed by the logout transaction. You should also consider to enable the loop option *Control the rate of iterations*. In the loop we have our transactions and one custom code element (Example 11-1). The job of this custom class is to generate a random number between 1 and 3. Depending on the result of the custom code, one of the three If statements is executed.

Example 11-1 Random selector custom code

```

package custom;
import java.util.Random;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/** @author André Kofaldt */
public class RandomSelector implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    private static Random random = new Random();

    /** Instances of this will be created using the no-arg constructor. */
    public RandomSelector() { }

    public String exec(ITestExecutionServices tes, String[] args) {
        return String.valueOf(random.nextInt(3) + 1);
    }
}

```

11.2.7 Tuning the operating system

On a default Windows installation, it is possible to simulate about 100 virtual users. Even with gigabytes of RAM this limitation still exists. The exact number highly depends on the performance schedule to be executed. The reason for this limitation is the SAP GUI library that is used by the RPT SAP extension. Although the library works perfectly for a single user, there are some limitations for multi-user tests, because the SAP GUI is created for each simulated user. Even if the GUI is not visible, it still consumes a lot of resources.

Especially for long test runs, you should change two TCP/IP parameters in the Windows registry (Figure 11-16). In decimal the value is respectively 65000 and 1000. The default values are 5000 and 4.

```
[HKLM\System\CurrentControlSet\Services\Tcpip\Parameters]
"MaxUserPort"=dword:0000fde8
```

```
[HKLM\System\CurrentControlSet\Services\Tcpip\Parameters]
"MaxFreeTWTcbs"=dword:000003e8
```

Figure 11-16 Change registry values for long test runs

A detailed description of the two registry values can be found here:

- ▶ MaxUserPort:

<http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/58791.mspx>

- ▶ MaxFreeTWTcbs:

<http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/regentry/94179.mspx>

11.2.8 Tips and tricks

This section covers the two topics: best practices and common problems. First we discuss the best practices. When you are familiar with performance testing the SAP GUI for Windows application, you can take this also as a short check list for new performance tests. After this, we cover some common problems that might appear during SAP GUI performance tests.

Best practices

Here are some best practices for SAP testing:

- ▶ Verify the prerequisites. In particular, verify that the SAP GUI for Windows version, including the patch level, is supported by RPT (“Supported environments” on page 386).
- ▶ Enable scripting on the SAP server and on every client (“Enable scripting” on page 387).
- ▶ Record single transactions (“Recording SAP GUI tests” on page 389).
- ▶ Create complex schedules (“Creating complex SAP GUI tests” on page 397).
- ▶ Verify SAP playback options before you start heavy load tests.
- ▶ Remember that every out-of-the-box agent can simulate only about 100 virtual users.

Common problems

Common problems with SAP testing include:

- ▶ After starting a recording of a SAP test, you get the following error dialog (Figure 11-17) claiming: Connection is disabled by SAP Server.



Figure 11-17 Connection is disabled by SAP server

Enable scripting by running the rz11 transaction on the SAP server as described in “Enable scripting” on page 387.

- ▶ Recording SAP GUI tests using the remote desktop fails.

The IBM Rational Test Agent starts the SAP GUI, not on your machine, but on the console of the remote workstation. The agent service must be stopped and started manually using a batch file as shown here:

```
set RASERVER_HOME=C:\Program Files\IBM\SDP70Shared\AgentController  
cd "%RASERVER_HOME%"  
cd bin  
ACServer.exe
```



Testing Citrix application environments

The IBM Rational Performance Tester Extension for Citrix Presentation Server lets you emulate and test the performance and scalability of applications deployed through Citrix-based server environments.

Features include:

- ▶ Simple wizard-based test recorder
- ▶ Ability to annotate test recording with automatic screen captures or comments
- ▶ Test scheduling, execution, and results analysis integrated in one solution
- ▶ Performance reports tailored for Citrix scalability tests
- ▶ Test recorder and test suite editor optimized for Citrix-based applications
- ▶ Ability to record ICA protocol and key mouse and keyboard events to simulate real users

The IBM Rational Performance Tester Extension for Citrix Presentation Server requires a license key.

12.1 Introduction

Citrix Presentation Server (formerly Citrix MetaFrame) is a remote access and application publishing product that allows users to connect to applications available from central servers. One advantage of publishing applications using Presentation Server is that it lets people connect to these applications remotely, from anywhere around the world, with minimum investment and setup. From the end users' perspective, users log in using the Citrix Presentation Client to their corporate network in a secure environment without the need to install any business applications on their own desktops.

Centralized management of the system significantly simplifies deployment effort and management of applications, whether it is for day-to-day operations or business critical applications.

12.2 Citrix Windows environment

Citrix Presentation Server is built on the Independent Computing Architecture (ICA), Citrix systems' thin client protocol. The Microsoft Remote Desktop Protocol, part of Microsoft's Terminal Services, is based on Citrix technology. The Citrix Presentation Server clients (ICA Clients) are available for several operating systems, including Microsoft Windows, Mac, OS, Linux, and other UNIX-like systems. The testing described in this book focuses on the Windows environment only.

Networks that use this software are reminiscent of the mainframe-terminal system, where a powered server does most of the processing work (that is, calculations, searches, or analyses) while smaller, less powerful machines provide the user interface.

A key challenge of such an architecture is performance: A graphically intensive application (as most are when presented using a GUI) being served over a slow network connection requires considerable compression and optimization to render the application usable by the client.

The Citrix performance tests provide important insight to such considerations.

12.3 Testing Citrix for Windows

Essentially, testing Citrix for Windows involves testing of the Citrix environment as a terminal and all of its related interactions initiated by the end user, against the Citrix Presentation Server. Such operations can include any end user interactions with a browser, opening/closing documents, or a complete workflow of a business application within the context of a Terminal session.

Since the ICA Client communicates with the Citrix Presentation Server at a very low level (mouse movements, key presses), the types of applications under test are protocol independent. It does not matter if the application is a client-server application, Java application, or simple HTML Web application. They all involve mouse movements and key presses when being recorded in a test script.

The Citrix system administrator manages the selection of the available software and business applications in the Citrix end user environment through Windows profiles and accounts. This Citrix environment to which the end user login, can be configured uniquely for each user.

Because the Citrix performance tests interact with the ICA Client at a very low level (mouse movements, key presses) any changes made to the test after the recording, such as moving test elements, adding loops or conditions, or inserting new sequences, can break the context of the emulated user actions and cause synchronization time-outs. It is essential to be aware of the context of user actions when you are editing the test.

12.3.1 Limitations

Citrix performance tests use window creation and change events, and optionally image recognition techniques, to synchronize user input with server output.

Before you record a session with a Citrix application, the behavior of that application must be perfectly reproducible. Specifically, the application must always create windows and GUI elements at the same locations and in the same sequence. Mouse or keyboard events must always produce the same output.

Consider these examples:

- ▶ If the application contains dialog boxes that run only on the first execution of a particular program or feature, such as tips or security warnings, ensure that they are disabled when you record the test. Any windows or dialog boxes that were recorded but are not displayed on subsequent executions, or displayed at different coordinates on the screen, will break the test and cause synchronization time-outs.

- ▶ If you save a file during a recorded session, the application might issue a warning for an existing filename when you replay the tests. If the warning was not in the recorded session, this will break the test and cause errors.

12.3.2 Important guidelines

The test case of a Citrix performance test must have perfectly reproducible behavior in a very precise manner. Specifically, the application must always create windows and GUI elements at the same locations and in the same sequence. Mouse or keyboard events must always produce the same output. Without these requirements fulfilled, the Citrix performance tests can be challenging to keep under control. RPT V7 is improved to address this known issue.

These guidelines should help you record a reliable test and avoid experiencing synchronization time-outs during test execution. Ensure that you have a working Citrix client environment and that you are able to connect to a Citrix server.

Ensure that the session that you are recording will be reproducible. To record tests that can be reliably replayed, follow these guidelines:

- ▶ If you save a file during a recorded session, when replaying the tests, some applications might produce a warning for an existing filename. If the warning was not in the recorded session, this might break the test and cause synchronization time-outs.
- ▶ If the applications that you are testing contain dialog boxes that run only on the first execution of a particular program or feature, such as tips or security warnings, ensure that they are disabled when you record the test. Any windows or dialog boxes that were recorded but do not reappear on subsequent executions might break the test and cause errors.
- ▶ Use dedicated test user accounts for performance tests. Ensure that the user accounts have minimal potential to cause problems if unpredictable mouse events occur outside of the application window.
- ▶ Set up the test accounts and applications to minimize unpredictable window events, such as new mail notifications, automatic updates, or daily tips. Disable extensible menus and hover text tool tips when possible.
- ▶ If possible, avoid using the *Start* menu to launch applications. Use the Quick Launch bar, desktop shortcuts, or select *Start* → *Run* and enter the name of the application.
- ▶ Do not launch applications or open files from locations that are likely to change, such as *Favorites* or *Recent Files*.

- ▶ When using cascading menus such as the *Start* menu, always wait for a moment for the submenu to display. After the recording, when editing the test, look at the mouse sequences that were generated to ensure that they follow the correct path to display the submenu.
- ▶ Do not rely on the *Recent Documents* menus to open files or launch applications, because these items are likely to change from one execution to another.
- ▶ When recording tests, before interacting with a window or dialog box, click the element to ensure that it gets focus, and then provide input.
- ▶ Ensure that mouse movements and keyboard events are clearly defined and relatively slow. If hover help (tool tips) or mouse hover actions are expected, be sure to wait sufficiently before moving on.
- ▶ After recording a session, some applications require user input before quitting (for example, to record any changes). This can cause discrepancies between the state of the application at the end of a session and at the beginning of a test execution. To avoid problems, at the end of a recording session, close all applications manually and cleanly end the session from the *Start → Log Off* menu rather than clicking *Stop* or *Close* on the Citrix client or the Citrix Recorder Control.

After recording, and while you edit the test, it is important to perform regular verification runs in order to validate the test with a single user. After each run, open the test execution history to make sure that the test synchronizes correctly. If necessary, change the synchronization level from *Conditional* to *Optional* on window events or image synchronizations that produce unnecessary time-outs. Only deploy the test on virtual users or run it in a schedule when the test is robust enough to run flawlessly with a single user.

12.4 Recording Citrix tests

In this section we describe how to record Citrix tests.

12.4.1 Prerequisites

Before you can test the performance of Citrix applications, the ICA Client must be installed on the same computer as IBM Rational Performance Tester. The ICA Client is required for recording and execution of performance tests. If you are deploying tests over remote hosts to emulate a large number of virtual users, the following software must be installed on each remote computer:

- ▶ ICA Client
- ▶ IBM Agent Controller

All IBM Agent Controllers must be of the same RPT version with the same version of ICA Client installed. The ICA Client can be downloaded free from the Citrix Web site:

<http://www.citrix.com>

Use the RPT Software Updater to install the latest version of RPT Version 7 with the Citrix extension and contact IBM product support for further instructions.

12.4.2 Recording a Citrix test script

You must record a Citrix session in RPT with the ICA Client. When you record, the recording wizard in RPT automatically starts the ICA Client and configures it for recording. When you have finished recording the session, the wizard generates a Citrix performance test.

Preferences

The behavior of the recording wizard is controlled by recorder preferences. To inspect the current settings, select *Window → Preferences*, and expand *Test → Citrix Recorder*. This procedure assumes that the defaults are set (Figure 12-1).

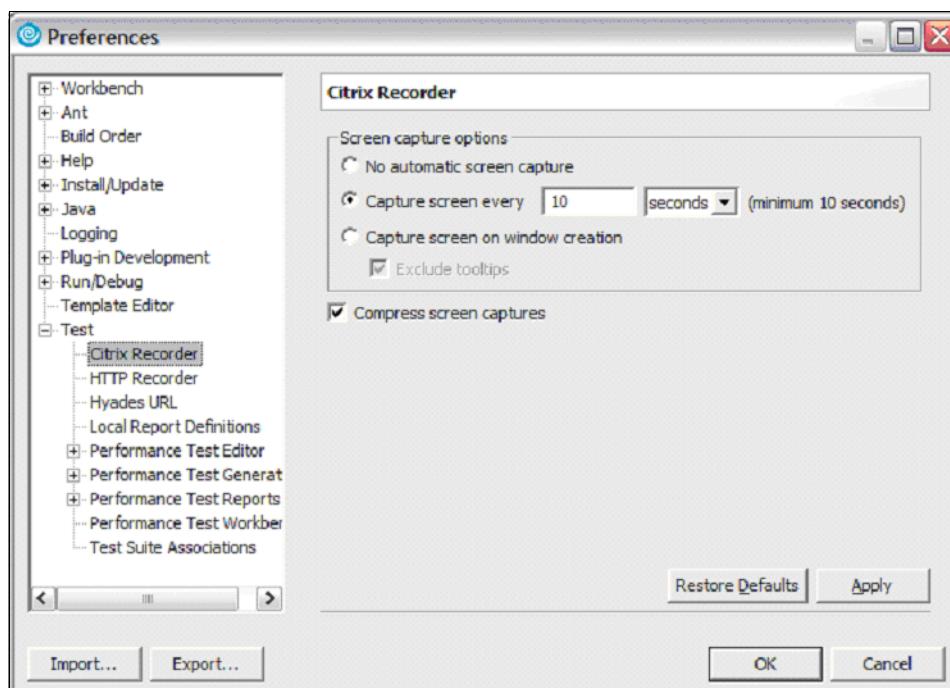


Figure 12-1 Citric recorder preferences

Recording a sample test

Tests are stored in Performance Test projects (Java projects that include a source folder).

To record a sample test:

- ▶ Select *File → New → Record Performance Test* and select *Citrix Performance Test*. Click *Next* (Figure 12-2).

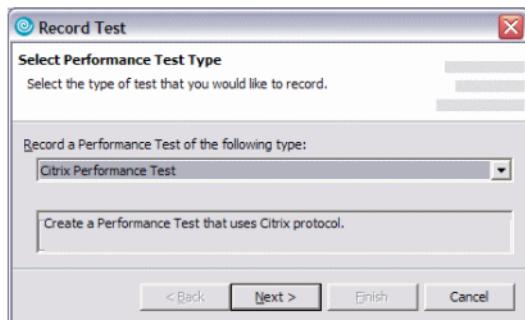


Figure 12-2 Recording a Citrix performance test

- ▶ On the Citrix Recording page, select a project. For Recording file name, type a name for the test, and click *Next* (Figure 12-3).

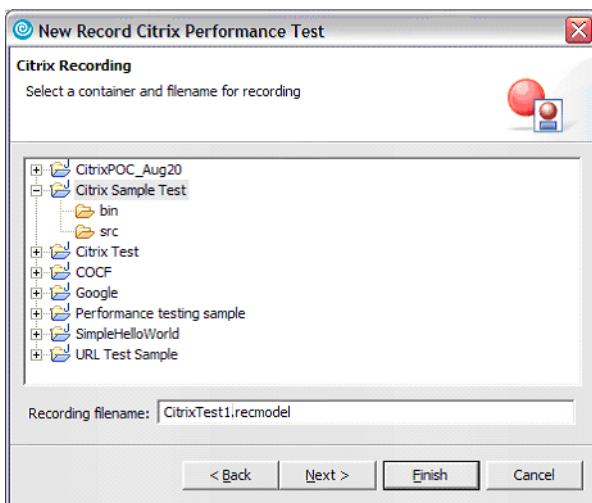


Figure 12-3 Citrix performance test: Name

The name that you type is the base name for the recording, test, and other required files. You see these files in standard Navigator or the Java Package Explorer with their distinguishing suffixes, but you see only the simple name (test) in the Test Navigator (for example, Citrix Sample Test is the Project Name and CitrixTest1 is the script name).

- ▶ On the Citrix Connection Settings page, specify how you want to connect to the Citrix server (Figure 12-4).
 - Select *On server* to connect directly to a specified Citrix server. Specify the name or IP address of the server (`http://CITRIXSERVER01`) or click *Browse* to locate a server on the local network. If the Citrix administrator has defined published applications, you can click *Browse* to select the application from the list of published applications on the server. To record a Citrix desktop session, leave this field blank. Click *Next* to continue.
 - Select *With ICA file* to connect using a predefined ICA file that is provided by the Citrix server administrator. Click *Browse* to locate and select the ICA file on your system and click *Next* to continue.

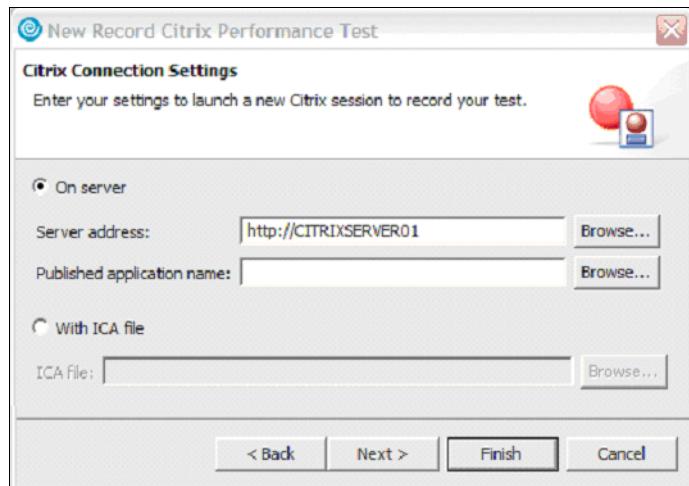


Figure 12-4 Citrix performance test: Server connection

- ▶ On the Citrix Session Options page (Figure 12-5), you can provide a description for the session and change the video settings for the ICA Client. Click *Next* to continue. Because Citrix performance tests are based on low level interactions with the server, including mouse and window coordinates, the Citrix session must be big enough to allow all actions.

You can do this by using a lower resolution for the recorded Citrix session than what you currently used for your own desktop. This will keep the window of the Citrix session to appear in full screen on your desktop without you having to use the horizontal and vertical scroll bars. The additional steps in clicking on the scroll bars can complicate the script unnecessarily.

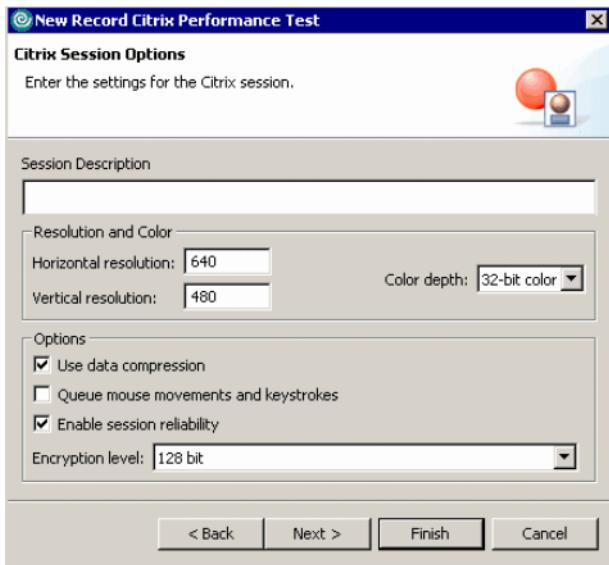


Figure 12-5 Citrix performance test: Options

- ▶ If this is the first time you record a Citrix performance test, read the Privacy Warning and select *I accept* to proceed.
- ▶ To start the recording, click *Finish*. A progress window opens while the ICA Client starts (Figure 12-6).



Figure 12-6 Citrix performance test: Progress

- ▶ A Citrix Recorder Control Window opens at the bottom right hand corner on the desktop, outside of the recording Citrix session (Figure 12-7). When *Display screen capture preview* is selected, the Preview will show screen captures every 10 seconds (or any other number) as set in the Citrix Recorder in *Window → Preferences → Test → Citrix Recorder*. This Citrix Recorder Control Window stays active throughout the recording session on your desktop to allow you to stay in control.



Figure 12-7 Citrix performance test: Recorder control

- ▶ Login to the Citrix server and perform the tasks that you want to test. You can use the Citrix Recorder Control window to add comments or take screen captures during the recording session to make meaningful indicators of what you are trying to achieve in a test case.

Sample use case

Consider this sample use case:

- ▶ After entering the logon ID and password, you are engaged in a Citrix session that looks exactly like an ordinary Windows desktop with pre-defined desktop setup. Click *OK* to dismiss the ICA Seamless Host Agent dialog box (Figure 12-8).

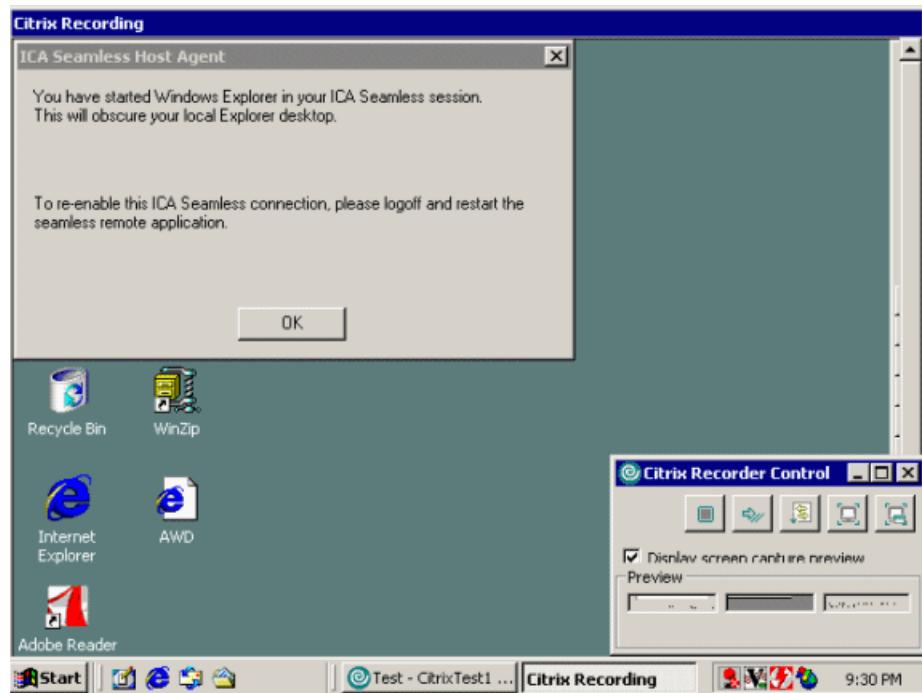


Figure 12-8 Citrix performance test: Session

- ▶ Open Adobe® Acrobat® Reader by double-clicking on the shortcut icon in Citrix at the bottom left hand corner.
- ▶ Click *Create Adobe PDF Online* once in Acrobat Reader. Microsoft Internet Explorer is launched automatically at the Adobe Web site (Figure 12-9). Close the browser. Acrobat Reader prompts if you want to do add a software update. Click *Cancel*.

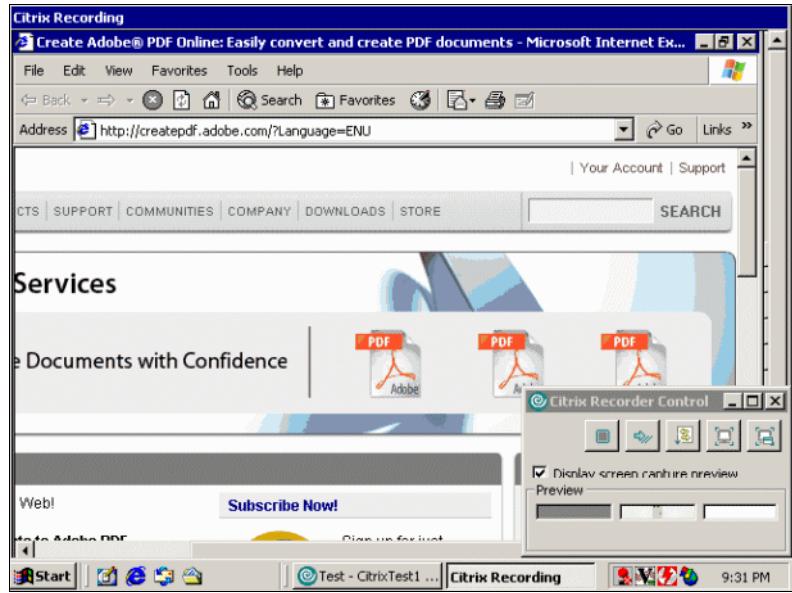
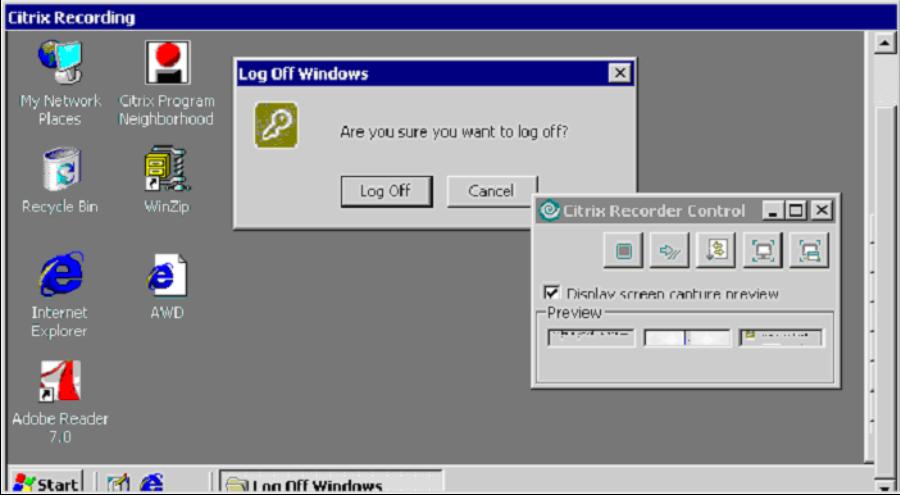


Figure 12-9 Citrix performance test: Adobe PDF

- One or more of the buttons shown in Table 12-1 in the Citrix Recorder Control box are used to help you capture a script for successful playback with verification points defined during the recording session.

Table 12-1 Citric recorder control buttons

| Icon | Usage of the Citrix Recorder Control Buttons during the Citrix recording session |
|------|--|
| | To add a user comment to the recorded test, click the <i>Insert user comment</i> button. Because Citrix tests can be long and difficult to read, meaningful comments can help you locate important elements. |
| | To add an image synchronization to the recorded test, click the <i>Insert image synchronization</i> button, select an area of the screen that will be used for synchronization, and click the <i>Insert image synchronization</i> button again. Image synchronizations allow the test to keep track of the contents of a screen area during the replay instead of focusing only on window events. You can use them to maintain synchronization of a test in applications that do not create or modify many windows, but update the contents of a window regularly. The contents of an image can be evaluated either as a bitmap hash code or as a text value obtained by optical character recognition. You can also add verification points to image synchronizations in the test editor. |

| Icon | Usage of the Citrix Recorder Control Buttons during the Citrix recording session |
|---|---|
|  | <p>To add a screen capture to the recorded test, click the Capture screen  icon. Screen captures make your tests easier to read and help you visualize the recorded test. To change the settings for screen captures, click Screen capture preferences  and select one of these options:</p> <ul style="list-style-type: none"> ▶ No automatic screen capture—Select this option if you do not want the test recorder to record screen captures automatically. When this option is selected, you can still record screen captures manually. This option is selected by default. ▶ Capture screen every specified period of time—Select this option to automatically record a periodic screen capture and specify the delay between captures. ▶ Capture screen on window creation—Select this option to record a screen capture each time a window object is created in Citrix. |

- ▶ Select *Start → Log Off Windows* (Figure 12-10).

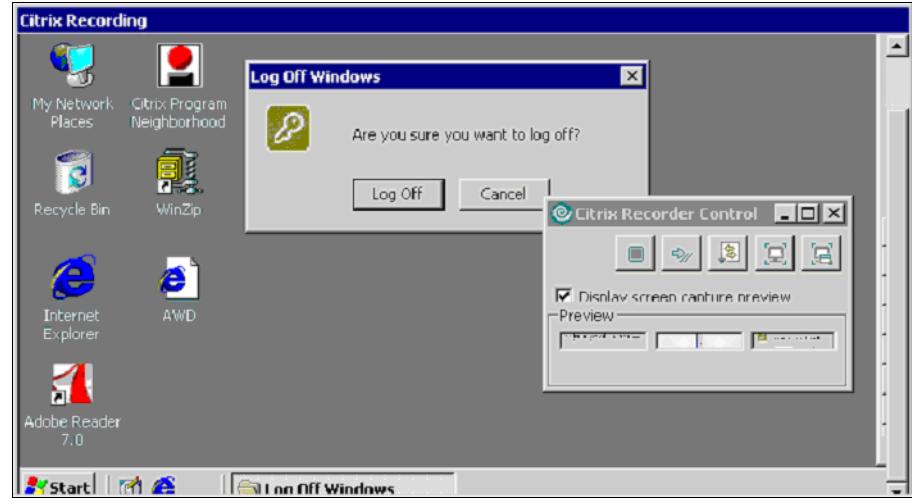


Figure 12-10 Citrix performance test: Logoff

- ▶ When you have completed the sequence of actions to be tested, close the session cleanly and stop the recorder by closing the ICA Client. A progress window opens while the test is generated. On completion, the Recorder Control view displays the message Test generation completed, the Test Navigator lists your test, and the test opens in the test editor.

A basic Citrix test script is created.

12.5 Editing Citrix tests

After you record a test, you can edit it to include variable data (rather than the data that you recorded), verification points (to confirm that the test runs as expected), transactions, conditional processing, and custom code.

12.5.1 Test editor

With the test editor, you can inspect or customize a test that you recorded to make it meaningful and descriptive. Without customization, the test is shown in Figure 12-11.

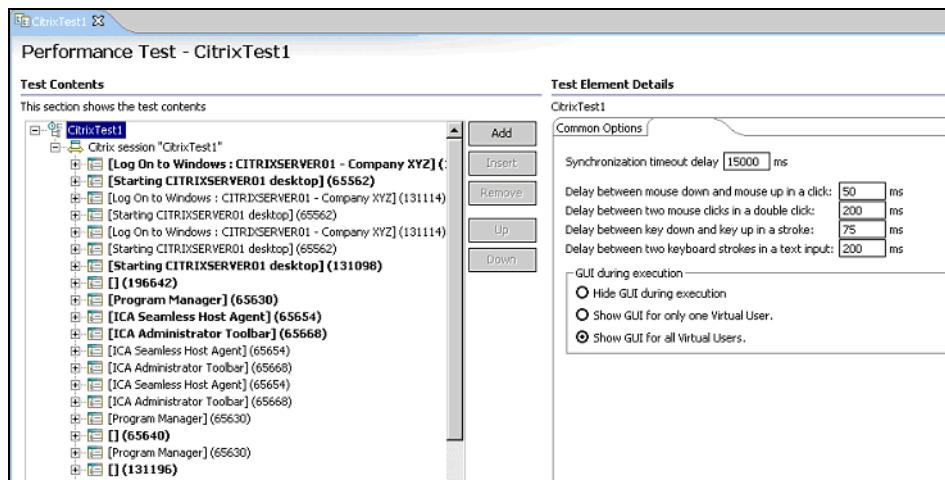


Figure 12-11 Test editor: Without customization

The test editor lists the window events for a test, in sequential order. New windows are displayed in bold. The Windows operating system assigns each window an ID number. This number changes on each execution of the test, but usually remains the same within the test, providing a means of identifying each window object.

Note: In some cases, the operating system recycles destroyed window IDs (for example, 65562). The test recorder identifies these properly by appending an extra number at the end of the window ID if necessary.

The example in “Recording Citrix tests” on page 405 shows the test of opening Acrobat Reader in a Citrix session, which was generated from a recording of these tester actions:

- ▶ Login to the Citrix server CITRIXSERVER01 (use server setup for your own environment).
- ▶ Start the Acrobat Reader program through the shortcut icon on the Citrix desktop.
- ▶ Click *Create Adobe PDF Online* in Acrobat Reader.
- ▶ Open the Web site of Adobe Information in Microsoft Internet Explorer automatically.
- ▶ Close the Acrobat Reader program.
- ▶ Click *Cancel* to update Adobe in software update.
- ▶ Log off Windows.

There are two main areas in the test editor window:

- ▶ The area on the left, Test Contents, displays the chronological sequence of events in the test.
- ▶ The area on the right, Test Element Details, displays details about the currently selected item (window, mouse event, key event, or screen capture) in the test hierarchy.

In the preceding example, Test Element Details displays information about the test because the name of the test, Paint, is selected in the Test Contents area. The Common Options and Citrix Options apply to the entire test.

The test editor lists the window events for a test, in sequential order (Figure 12-12). New windows are displayed in bold. The Windows operating system assigns each window an ID number. This number changes on each execution of the test, but usually remains the same within the test, providing a means of identifying each window object.

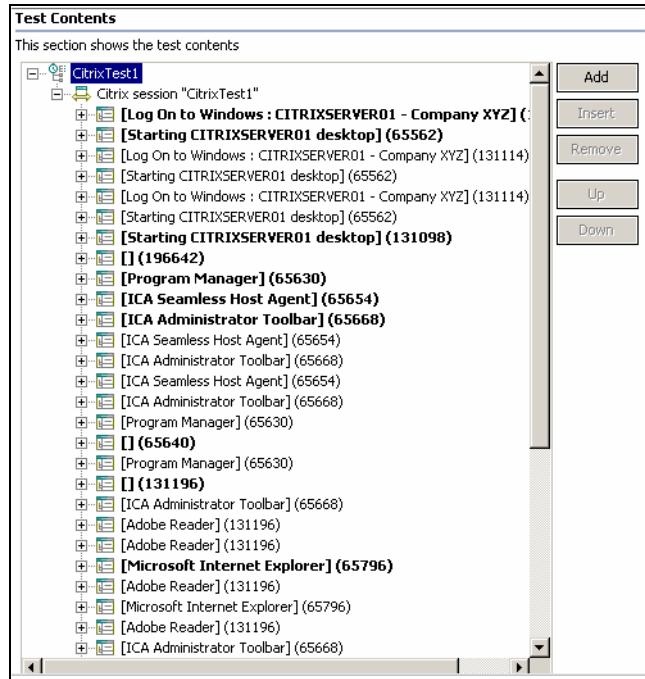


Figure 12-12 Test editor: Test Contents

12.5.2 Test element details

The Common Options tab in the Test Element Details (Figure 12-13) displays the preset recording parameters in *Windows → Preferences → Expand on Test → Performance Test Generation → Citrix Test Generation*. Select *Default Test Execution Delays*.

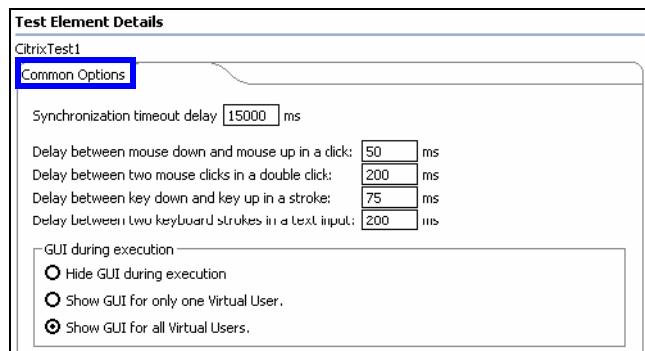


Figure 12-13 Test editor: Test Element Details Common Options

12.5.3 Citrix session

Under the test is the Citrix session, which contains information about the connection and ICA Client options, such as color depth and resolution (Figure 12-14).

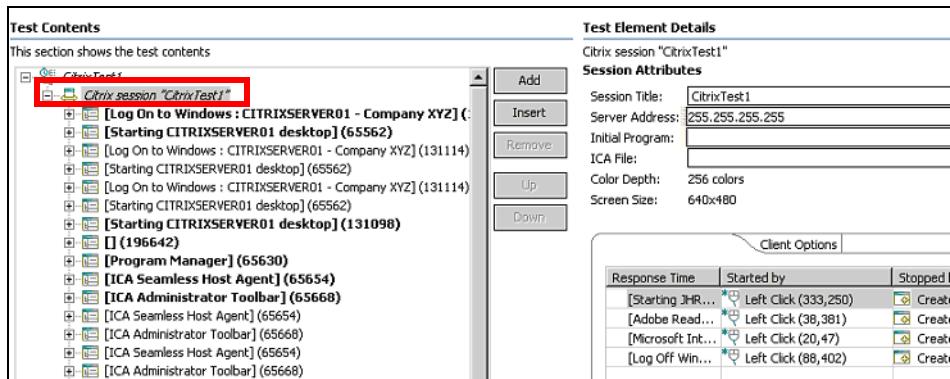


Figure 12-14 Test editor: Citrix session

12.5.4 Window events

Window events are the primary test elements in a Citrix test and represent graphic objects that are drawn by the Citrix server, such as actual window, dialog boxes, menus, or tool tips. A Window event is recorded each time a window is created, destroyed, moved, or resized. The first occurrence of a window, a create window event, is displayed in bold. Window objects are typically identified by their title. If there is no window title [], for example, on menus or tool tips, then the test editor uses the window ID number.

Inside windows, you see a list of events for the window, such as create window events, screen captures, and mouse or keyboard actions (Figure 12-15).

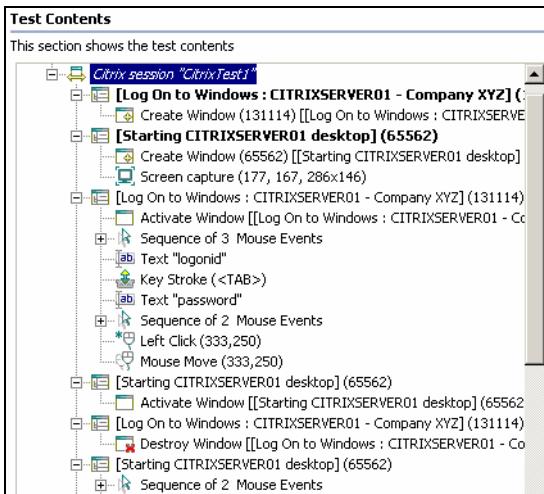


Figure 12-15 Test editor: Windows events

Some actions contain data that is highlighted. This highlighting indicates that the data contains one or both of the following types of information:

- ▶ **Datapool candidate**—This is a value, usually one specified by the tester during recording, that the test generator determined is likely to be replaced by values in a datapool. An example of a datapool candidate is a string that you search for in a recorded test. The string is highlighted as a datapool candidate on the assumption that, before running the test, you might want to associate the string with a datapool column containing appropriate substitute values.
- ▶ **Correlated data**—These are values in a test, usually one of them in a response and the other in a subsequent request that the test generator determined needed to be associated in order to ensure correct test execution. An example is a photograph returned to the browser by a test that searches an employee database. The test generator automatically correlates employee names with photographs. Suppose that, before running the test with many virtual users, you replace the employee name searched for in the recorded test with names in a datapool. Because the test correlates the data, each virtual user searches for a different employee, and the server returns an appropriate photograph.

To see an illustration of color coding in performance tests, select *Window → Preferences → Test → Performance Test Editor*, and then select the *FONTs and Colors* tab.

Click *Add* to add elements to the selected test element. Alternatively, you can right-click a test element and select an action from a menu.

The choices that you see depend on what you have selected. For example, inside a window, you can add a mouse action or a text input. The *Insert* button works similarly. Use it to insert an element before the selected element. The *Remove* button allows you to delete an item.

Note: Because Citrix performance tests rely on low level interaction with the server, manually changing test elements is likely to break a recorded test.

Sometimes, the area of the editor where you have to work is obscured. To enlarge an area, move your cursor over one of the blue lines until your cursor changes shape (to a vertical line with an up arrow at the top and a down arrow at the bottom) and drag up or down while holding the left mouse button.

12.5.5 Screen capture

A screen capture at the point of recording is shown in Figure 12-16.

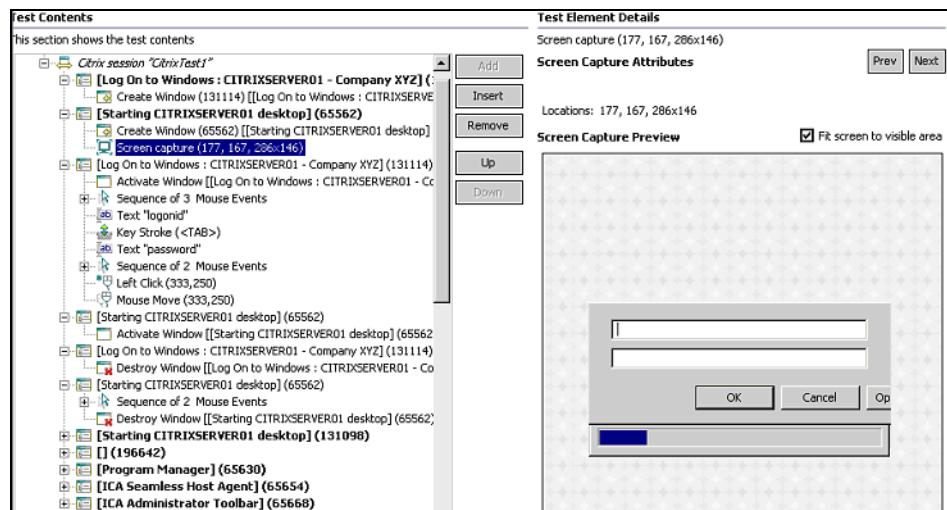


Figure 12-16 Test editor: Screen capture

You can have as many screen captures as you want to track along specific steps in a specific test case. The screen captures do not play a functional role in the playback. A screen capture is particularly useful to help you identify the step with no Windows Title but only a window ID. For example, Screen capture of [] (65632) indicates that start menu is created in the Program Manager (Figure 12-17).

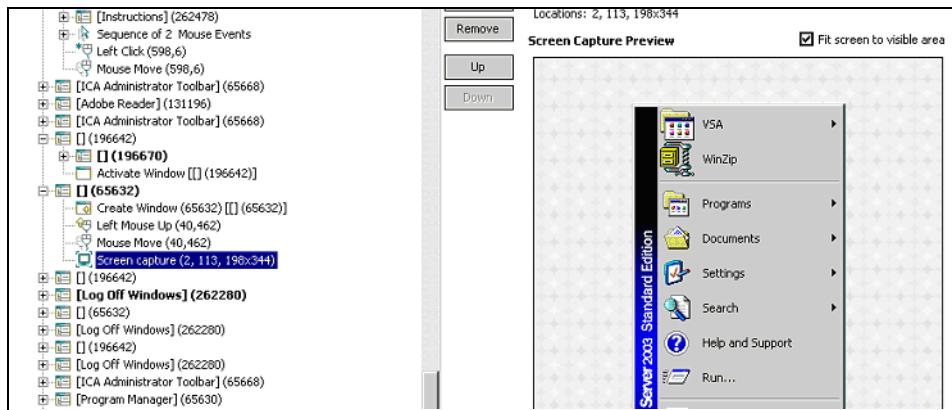


Figure 12-17 Test Editor: Screen capture

12.5.6 Image synchronization

You insert image synchronizations during the recording phase. These images are specific to your Citrix application.

Examples of image synchronization are shown in Figure 12-18.



Figure 12-18 Examples of image synchronization

12.6 Evaluating Citrix test runs

Specifically for measuring Citrix, there are the Citrix performance report and the Citrix verification point report.

They are both in some way similar to their HTTP Report counterparts.

12.6.1 Citrix performance report

Different from the HTTP Overall result, the Citrix Overall result records the following information:

- ▶ A progress indicator that shows the state of the run.

- ▶ A bar chart that shows indicates the overall success of the run with the percentage of window synchronization successes and the percentage of image synchronization success. Synchronization success indicates that the expected window events in the test match the actual window events in the test run.
- ▶ A bar chart that shows indicates the overall success of the run with the percentage of image synchronization successes and the percentage of image synchronization success. Synchronization success indicates that the expected image area or extracted text in the test matches the actual image area or extracted text in the test run.

The Citrix Summary section displays the following information:

- ▶ The average response time for all response time measurements. Response times are determined by measurements that are located in the tests. Response time measurements can be automatically generated between the last input action before a window create event and the window create event. The table does not display values that equal zero.
- ▶ The total number of image synchronization attempts.
- ▶ The total number of image synchronization successes.
- ▶ The total number of image synchronization time-outs. A time-out occurs when the synchronization fails.
- ▶ The total number of window synchronization attempts.
- ▶ The total number of window synchronization successes.
- ▶ The total number of window synchronization time-outs. A time-out occurs when the synchronization fails.
- ▶ The maximum response time for all response time measurements. This indicates the highest response time that was measured during the run.
- ▶ The minimum response time for all response time measurements. This indicates the lowest response time that was measured during the run.
- ▶ Total user actions for run. This indicates the total number of user input actions that were emulated during the run.

In the Citrix Summary example in Figure 12-19:

- ▶ There is 50% (47 divided by 94) success rate of finding the expected content (within the Citrix session) through Citrix image synchronization.
- ▶ There are 66% (319 divided by 485) success rate of creating, activating and destroying expected window events.
- ▶ There are 162 Citrix Window time-outs.

| Citrix Summary | |
|---|------------|
| Average Response Time For All Timers For Run [ms] | 67,668.021 |
| Citrix Error Code | 203 |
| Citrix Image Synchronization Attempts | 94 |
| Citrix Image Synchronization Successes | 47 |
| Citrix Image Timeouts | 47 |
| Citrix Window Synchronization Attempts | 485 |
| Citrix Window Synchronization Successes | 319 |
| Citrix Window Timeouts | 162 |
| Maximum Response Time For All Timers For Run [ms] | 1,227,886 |
| Minimum Response Time For All Timers For Run [ms] | 0 |
| Total User Actions For Run | 4,254 |

Figure 12-19 Citric performance report: Summary

This is not considered a highly successful test. An ideal test should have the highest possible successful rate of the aforementioned figures.

The Server Timeout tab provides page shows when the synchronization time-outs and server errors occurred during the run. The graph does not display values that equal zero.

The line graph shows the following information:

- ▶ Citrix window synchronization time-outs
- ▶ Citrix image synchronization time-outs
- ▶ Citrix server errors or errors encountered during test execution

12.6.2 Citrix verification point report

The Citrix Image Synchronization Verification Point Summary table lists the following information for a run:

- ▶ The percentage of image synchronization verification points that passed
- ▶ The number of image synchronization verification points that were tested
- ▶ The number of image synchronization verification points that passed
- ▶ The number of image synchronization verification points that failed

The Citrix Verification Points page contains tables with verification point details. There are two types of Verification point details: Windows verification points and Citrix Image Synchronization verification points.

Windows verification points

The Windows Verification Points table lists the number of Windows verification points that passed, failed, caused an error, or were inconclusive, and the percentage of Windows verification points that passed.

Figure 12-20 shows an example of good Windows activities with 100% success rate of synchronization.

Window VPs

| | Pass -- Count [for Run] | Fail -- Count [for Run] | Verification Points -- Percent Pass |
|-----------------------------------|------------------------------------|------------------------------------|--|
| [ICA Seamless Host Agent] (65652) | 32 | 0 | 100 |
| [Program Manager] (65628) | 32 | 0 | 100 |
| [Log Off Windows] (196900) | 16 | 0 | 100 |

Figure 12-20 Windows verification points: Good result

Figure 12-21 is an example of bad Windows synchronization with a 100% failure rate. It is shown at the end of the table. It means that the simulation of these activities missed the targets. These errors can be reduced with your effort to fine tune the test scenario.

| | | |
|---|----|---|
| [Microsoft Internet Explorer displays AWD content on page] (196726) | 26 | 0 |
| [Run IE to launch AWD. html pointing to DAS1] (131190) | 27 | 0 |
| [Screen capture of User ID found on AWD Home page] (131496) | 26 | 0 |
| [Click on Program Manager Run] (65630) | 31 | 0 |
| [AWD Content Viewer launched] (459294) | 23 | 0 |

Figure 12-21 Windows verification points: Bad result

Image synchronization verification points

The Image Synchronization Verification Points table lists the number of image synchronization verification points that passed, failed, caused an error, or were inconclusive, and the percentage of image synchronization verification points that passed.

You record the image synchronization you wanted while you are recording the test. These images are specific to your Citrix application, that is, it can be an image of a button, or an area of a screen shot.

Figure 12-22 shows examples of the image synchronization. This is not considered a highly successful test with regards to image synchronization verification points. An ideal test should have the highest possible successful rate.

| Image Synchronization VPs | | | |
|--|----------------------------|----------------------------|---|
| | Pass -- Count [for Run] | Fail -- Count [for Run] | Image Synchronizat ion Points -- Percent Pass |
| Image synchronization (30, 287, 54x23) | 19 | 7 | 73.077 |
| Image synchronization (21, 176, 42x23) | 22 | 1 | 95.652 |
| Image synchronization (321, 298, 134x48) | 3 | 20 | 13.043 |

Figure 12-22 Image synchronization verification points

12.7 Creating complex Citrix tests

Because each Citrix application can have a different level of testing challenges in terms of the amount of key strokes, screen shots, image for synchronization to manage, you are actually the person who has to determine the best possible successful rate: The higher the better.

These errors are, by nature, common challenges to Citrix testing. The iterative tuning of the test scenario helps reducing some of these errors, but unfortunately these errors cannot be totally eliminated. There is also no proven technique to clearly distinguish *Test errors because of tool* from the *Real Citrix server errors*, because of the low level of Citrix testing, which—apart from the synchronization and verification points—should provide some clue.

Synchronization is used for Citrix testing and produces synchronization time-outs in the test execution when the test fails to recognize the server output.

Synchronization errors are not necessary server errors. Here is why:

- ▶ Synchronization occurs on window events or through the recognition of a screen area specified by the user. Consider the virtual server being like a blind-folded business user trying to do mouse clicking and keyboard commands onto the computer sending a sequence of user commands to the Citrix desktop. The simulated user relies heavily on the timing to be clicking the right screen area with the right menu showing up on the citrix desktop – to be able to carry out a business transaction. This is the reason why this kind of testing is challenging.

The ability to synchronize actions with what the server responses determine the amount of errors the test produce up to a point when synchronization errors can be eliminated no more—the true server errors are resulted, confirmed further by server side monitoring.

12.8 Optimizing Citrix tests

You might have to do a lot of tries before you can achieve an optimized, solid test.

If you notice that the Citrix playback is often slowed down by too many time-outs, there is a trick that you can apply to produce playback that is quicker, roughly all of the time, executed until the end.

The main concern with this algorithm is that it is implemented neither in version 6.1.2 nor in 7.0. The situation will be better in version 7.0 because it will be easy to deliver hot fixes.

Apply this trick manually within the script until there is a hot fix. This suggested method is just an example, it might not be applicable in all cases:

- ▶ Remove all ACTIVATE, RESIZE, MOVE window events. To find these events, it is best to use the *Text search* menu (Figure 12-23).

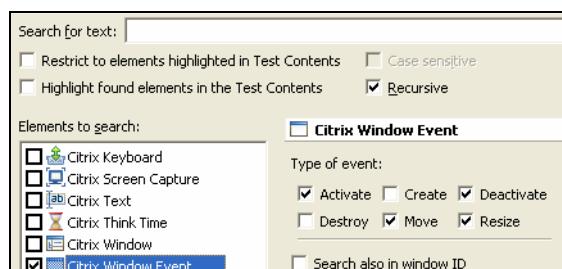


Figure 12-23 Searching for events

- ▶ Remove windows without events (Figure 12-24).



Figure 12-24 Remove windows without events

- ▶ Remove unuseful mouse move. Because a mouse click event has coordinates, mouse move are often unuseful. In general single mouse move can be removed without a problem.
- ▶ Gather key, text, and click when possible.

For example: you can optimize these key strokes (Figure 12-25).

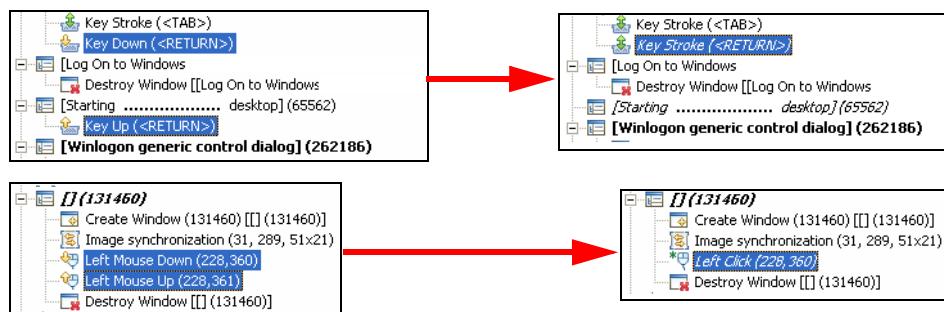


Figure 12-25 Optimizing key strokes

- ▶ Remove windows without playable events.

This step is a little more complicated but very efficient. A playable event is a key event or a mouse event. If you see that no occurrences of a window contain playable events, then you can remove all the occurrences of this window.

For example, following that method you could remove all the occurrences of:

- Starting CITRIX desktop (65562)
- Starting CITRIX desktop (131098)
- Winlogon generic control dialog (262186)
- Program Manager (65628)
- [] (65556)
- [] (65870)

- ▶ Check the test suite options and the response time definition.

You are now ready to run the test suite again and to check that the execution is still correct. You should get better results, because this can really improve playback.

Hints and tips: If you have multiple clicks with nothing between them, it means that you are clicking rapidly in repetition. It might be wise to not add a synchronization image within these steps.



Testing with custom Java code

This chapter describes Rational Performance Tester (RPT) custom code support and discusses some of the techniques for its use in application load or performance testing.

We cover the following topics:

- ▶ What RPT custom code is, and what it can be used for
- ▶ The basics for RPT custom code use
- ▶ Using custom code in RPT performance tests
- ▶ Synthetic workload simulation

Note: This chapter assumes a working knowledge of the Java programming language and basic RPT usage.

13.1 What is custom code?

Simply put, RPT custom code support allows RPT to execute a user's Java test code from within an RPT schedule, either by itself or in conjunction with other RPT-generated performance tests.

This support adds a flexible and powerful mechanism to the basic RPT application under test (AUT) record-and-playback (RAP) technique for test automation.

Among other things, custom code support allows an RPT user to:

- ▶ Extend and customize existing RPT-generated test behavior.
- ▶ Provide test coverage for applications that do not have Web clients or have no separate client application at all.
- ▶ Programmatically create *synthetic workload* tests (that is, tests that were not generated from previously recorded user actions) and then execute those tests.

13.2 Custom code usage

There are two modes for using RPT custom code support:

- ▶ Extend existing RPT performance testing
- ▶ Synthetic workload testing

Extending existing RPT performance testing

The first mode focuses on extending existing RPT RAP performance test behavior.

The following brief list provides examples of the extensions to RPT test behavior that can be implemented using custom code support:

- ▶ Logging a message to the RPT problem determination log, perhaps to assist in product or test debugging.
- ▶ Logging an event or message to the RPT test log, to provide additional test run context.
- ▶ Customizing the execution flow of an RPT schedule loop, perhaps to implement special test behavior.
- ▶ Providing customized product-specific test data processing or validation.
- ▶ Definition and control of custom AUT transactions.

- ▶ Use of custom RPT statistical counters for special product metrics collection and reporting.
- ▶ Passing (persisting) data from one test to another, for the same simulated user.

Synthetic workload testing

The second mode focuses on what can be considered as *synthetic workload* load or performance testing.

In this mode, RPT is not used to execute tests that were generated from previously recorded AUT user actions. Instead, RPT is used as an execution framework to execute Java test code that is layered on top of an AUT application program interface (API). The software architecture for this type of test is shown in Figure 13-1.

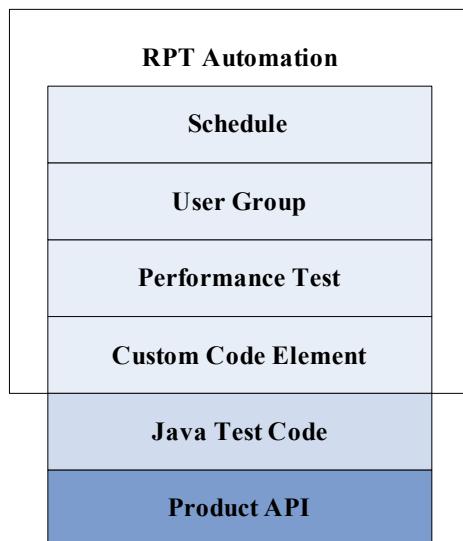


Figure 13-1 Test architecture using RPT custom code

Using this technique, the simulated user workload is programmatically, or *synthetically*, simulated for the application.

This testing approach can be useful if:

- ▶ The AUT has no Web client or no client at all (that is, the application provides an API that can be invoked from Java).

In this case, custom code serves as the *glue* between the application client/API and an RPT performance test.

- ▶ The application client provides intelligence that makes it difficult to model using a record-and-playback approach.

For example, the application client maintains a large amount of state locally stored on the machine where the client is running that makes its impractical to automate.

All of the extensions that can be made added to RPT performance tests using custom code can also be made to synthetic workload performance tests.

RPT also supports mixing the use of both custom code support modes in the same test schedule.

13.3 The basics of custom code

Java test code is incorporated into an RPT performance test through a custom code element. This custom code element provides the linkage needed by RPT to execute the Java test code within the context of the performance test and pass arguments to the Java test code executed via the custom code element.

This custom code element can be added to an existing performance test generated from a previously recorded client application session or can be added to a new, empty test.

13.3.1 Adding custom code

To add a custom code element to a performance test, open the test from the Test Navigator window, right-click on the test object in the Performance Test Contents window, and select *Add → Custom code* (Figure 13-2).

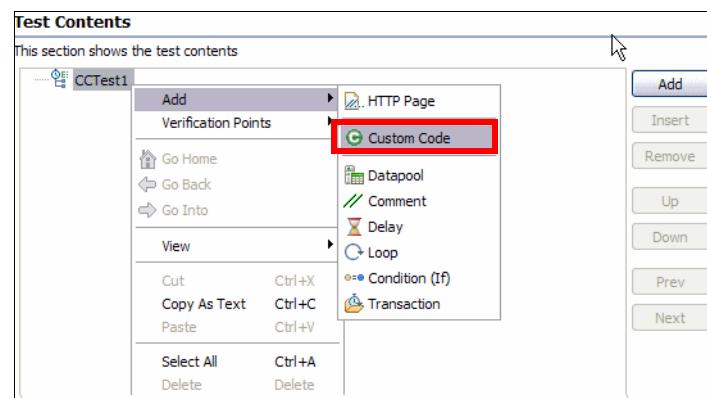


Figure 13-2 Adding a custom code element

This action inserts a new custom code element and displays a Test Element Details window for the new custom code object on the right (Figure 13-3).

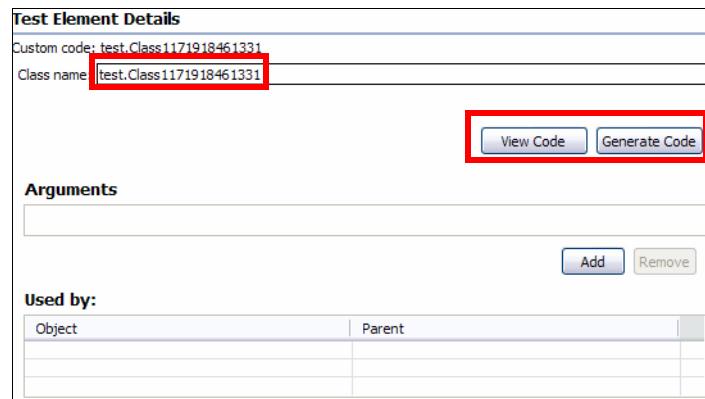


Figure 13-3 New custom code element

In the Test Element Details window on the right, follow these steps:

- ▶ Change the Class name to something related to the purpose of the Java test code to be associated with the custom code element.

Note that by default the class is part of the test package (Figure 13-4). This is the Java package that RPT uses for all its generated Java test classes.

We recommend that another Java package be used to avoid what can be a confusing inter-mingling of RPT and user Java test code.



Figure 13-4 Partial RPT test package Java class listing

- ▶ Click either:
 - *View Code* to add existing Java test code to the custom code element
 - *Generate Code* to have RPT generate a new skeleton Java test code class.

Warning: Be sure to click *View Code* to add existing Java test code; otherwise, *Generate Code* will overwrite existing Java test code.

Note: It is worthwhile to give some thought to a set of Java coding guidelines, including package and class name conventions. This effort will pay dividends over time, as they will make the Java test code easier to organize and maintain.

When either button is clicked, RPT either opens the existing Java class and displays its source code, or creates the corresponding skeleton Java class and displays the class source code (Figure 13-5).

This Java class was generated by RPT by clicking *Generate Code* in the previous example, and is part of the user's myPkg package.

```
package myPkg;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author unknown
 */
public class myValidation implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public myValidation() {
    }

    /**
     * For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select 'Help
     * Contents' in the Help menu and select 'IBM Rational Performance Tester TES'.
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        return null;
    }
}
```

Figure 13-5 Generated custom code element class

The previous example of custom code class satisfies two important requirements imposed by RPT on Java test code it is to execute:

- ▶ The Java test code must implement the ICustomCode2 interface.
- ▶ The Java test code must accept two arguments, an object of the ITestExecutionServices class and a String[] array, as its first and second arguments, respectively.

Both the ICustomCode2 interface and ITestExecutionServices class can be found in the com.ibm.rational.test.lt.kernel.services package. An import of this package is automatically included in RPT-generated Java class code.

The first requirement is important, in that implementation of the ICustomCode2 interface requires an implementation of the exec method. This is the method that an RPT custom code element calls to execute the user's Java class.

The second requirement serves to provide a mechanism by which the user's Java test code can gain access to support services, such as RPT logs (more on these details in 13.4, "Using ITestExecutionServices" on page 437).

At this point, the generated custom code class does nothing and simply returns null (Figure 13-5 on page 434).

13.3.2 An example of custom code use

The simple Java test code displayed in Figure 13-5 on page 434 should be changed or replaced with the necessary Java code to provide the desired functionality.

One example of such functionality is to provide custom validation of AUT run-time data. Figure 13-6 shows an example of custom AUT run-time data validation.

```

* For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select 'Help Contents' in the toolbar
public String exec(ITestExecutionServices tes, String[] args) {
    ITestLogManager itestLogManager = tes.getTestLogManager();

    IPDLogManager ipdLogManager = tes.getPDLogManager();
    IDataArea dataArea = tes.findDataArea(IDataArea.VIRTUALUSER);

    // Get current DevStream from data area and the response content from the CQWJ test
    int i = 0;
    String response = args[0];
    String devStream = (String) dataArea.get("sptCCRC_devStream");
    ipdLogManager.log(IPDLogManager.INFO, "In CC/CQ Integrations VerifyChangeSet module");

    // string manipulation, search for devStream
    int beginIndex = response.indexOf(devStream);
    if (beginIndex == -1) {
        ipdLogManager.log(IPDLogManager.SEVERE,
            "Didn't find the devStream in the Response content");
    }
    else { i = 1; // found the devStream in the response content }

    // check for the change view set
    String str = devStream+"\\";

    ipdLogManager.log(IPDLogManager.INFO, "devstream in ChangeSet" + str);

    int iIndex = response.indexOf(str);
    if (iIndex == -1) {
        ipdLogManager.log(IPDLogManager.SEVERE,
            "Didn't find the devStream in the response content");
    }

    if ((iIndex != -1) && (i == 1)) {
        // found changeset
        itestLogManager.reportVerificationPoint("Found DevStream in ViewChangeSet", VerdictEvent.VERDICT_PASS);
    } else
        itestLogManager.reportVerificationPoint("Didn't find ViewChangeSet", VerdictEvent.VERDICT_FAIL);
    return null;
}

```

Figure 13-6 Custom data validation example

This example has several important aspects:

- ▶ Use of the args [] array to pass input to the method for processing (covered in more detail in 13.6.1, “Passing data to custom code” on page 457).
- ▶ Use of an RPT data area to retrieve previously stored data for processing (covered in more detail in 13.4.2, “Data areas” on page 443).
- ▶ Use of the RPT problem determination (PD) log to record informational and test-error messages (covered in more detail in 13.4.1, “Logging support” on page 437).
- ▶ Use of the RPT test log to report test verdicts (also covered in more detail in 13.4.1, “Logging support” on page 437).

The args[0] string is used to pass a copy of the current AUT response buffer to undergo validation to the method in Figure 13-6. This validation is to search for a specific string, as specified by the sptCCRC_devStream value in the RPT IDataArea for the virtual user for the test that called this method. The IPDLogManager is used to record informational messages regarding the validation process and record unexpected errors encountered during validation processing. And finally, the ITestLogManager is used to report a test verdict based on the results of the data validation.

The support for three of these aspects in the previous example Java method is provided by the RPT ITestExecutionServices interfaces and classes. This support frequently plays a key role the implementation of the desired functionality of Java test code executed via RPT custom code support.

13.4 Using ITestExecutionServices

When a custom code element has been added to an RPT test, and that custom code element is revised to call Java test code, additional information is often needed for that test code to fully implement its functionality.

The Java test code might require:

- ▶ An RPT log to record product or test information.
- ▶ An RPT log to report test results.
- ▶ The total number of virtual users active for the RPT test schedule.
- ▶ The unique virtual user ID assigned to the RPT test that is currently running.
- ▶ The elapsed time since the start of the RPT test schedule.
- ▶ A location to store or retrieve test run-time information

These and a variety of other information and services are provided to Java test code through the RPT ITestExecutionServices interface and classes.

RPT on-line help provides a summary of this information and services at *Help Contents → Extending Rational Performance Tester Functionality → Extending Test execution with custom code → Test execution services interfaces and classes*.

In the following sections we examine this information and services more closely.

13.4.1 Logging support

RPT provides two logs for performance tests: One log is used for test debugging and problem determination, and a second log is used for test verdict reporting and failure information.

Problem determination log

The first type of log, called the *problem determination log*, is supported by the IPDLogManager interface.

This interface provides a variety of logging levels, and can be used to report test error conditions or problems, as well as report informational messages. As a result, the IPDLogManager interface is often used as a Java test code debugging tool for code executed by RPT.

Nine logging levels are supported: None, Severe, Warning, Info, Config, Fine, Finer, Finest, All. These levels range from least verbose to most, and control which messages instrumented in the Java test code will be recorded in the log.

The logging level selected for a test run is controlled by the RPT schedule for the test run. This schedule setting is shown in the Schedule Element Details (Figure 13-7).

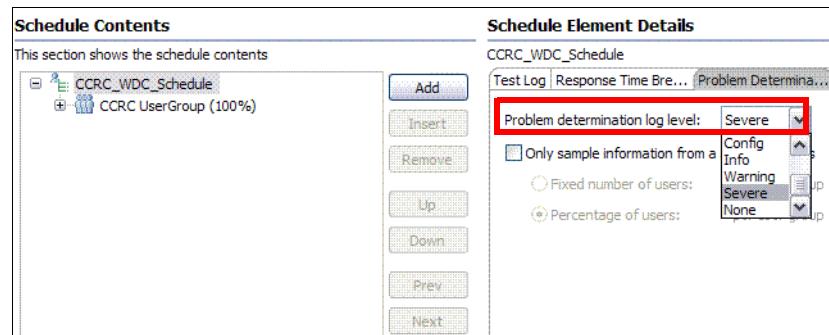


Figure 13-7 Selecting the problem determination log logging level

It is a best practice to conditionalize any information to be written to the problem determination log with a call to the `wouldLog` method of the IPDLogManager object, as in this sample code fragment:

```
IPDLogManager ipdLogManager = tes.getPDLogManager();
ipdLogManager.wouldLog(IPDLogManager.FINE) {
    ipdLogManager.log(IPDLogManager.FINE,
        "CustcomCode1: Inside Func(1)");
}
```

Where `tes` is an instance of `ITestExecutionServices`.

This code fragment writes `CustcomCode1: Inside Func(1)` to the problem determination, when the FINE log level is selected within the RPT schedule calling the fragment.

To view information written to the problem determination log, go to the Profiling and Logging perspective, and within the Log Navigator tab, click *Import* (Figure 13-8).

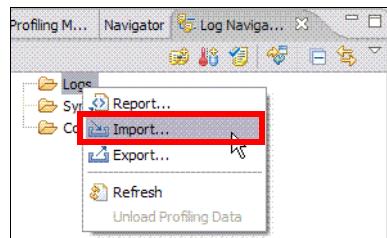


Figure 13-8 Starting to import a problem determination log

In the Select dialog window, select *Profiling and Logging* → *Log File* and click *Next* (Figure 13-9).

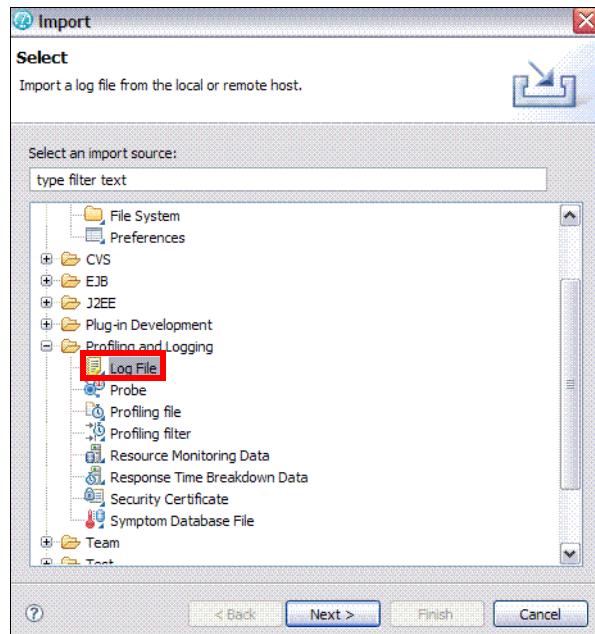


Figure 13-9 Selecting what artifact type to import

In the Log Files dialog, click *Add* (Figure 13-10).

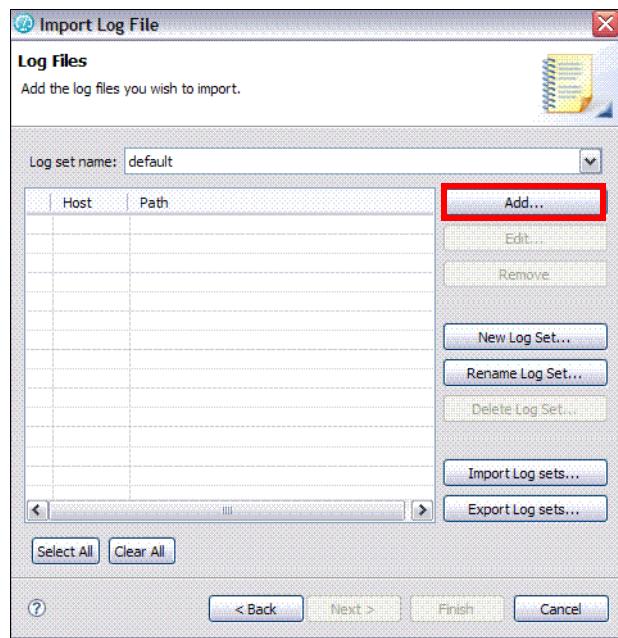


Figure 13-10 Adding a log file type to import

In the Log Types dialog, select *Common Base Event XML log*, and click *Browse* (Figure 13-11).

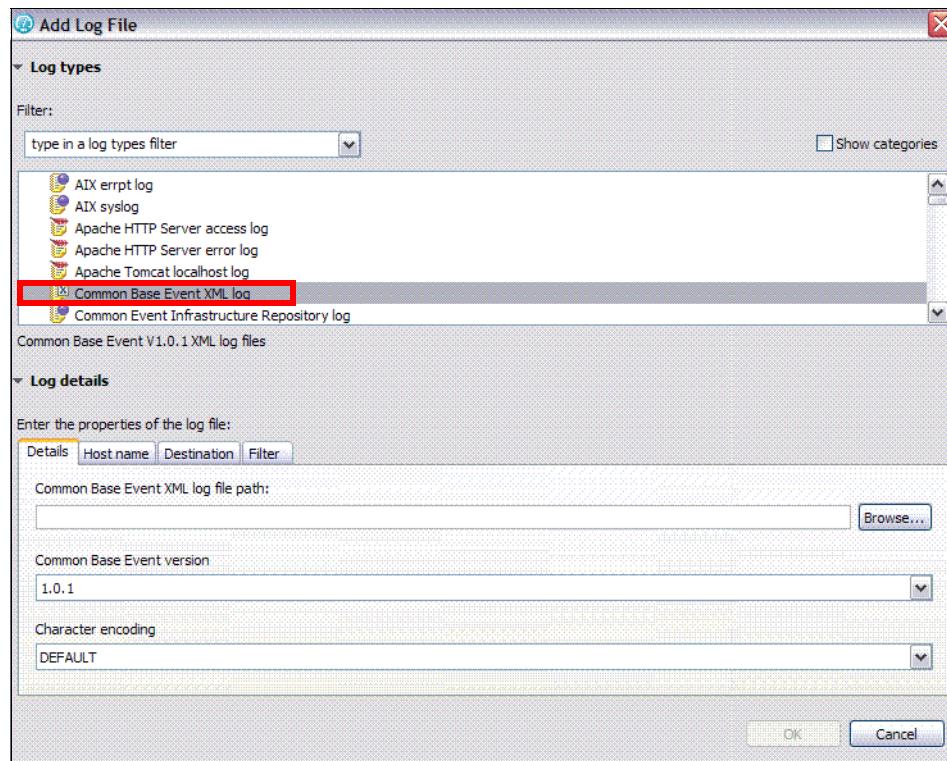


Figure 13-11 Adding a log file type to import

To locate the problem determination log to import, look in the RPT deployment directory for the desired machine. Then click *OK*, and the log is imported and available for review.

An example of an imported problem determination log is shown in Figure 13-12.

| Log View - Common Base Event XML log D:\WDC_RPT7.0_Workspace\deployment_root\testern\DC58C1C075E2C5041AA55F3... | | | | | |
|---|----------|--|----------|-----------------|---|
| Filter: No filter | | | | | |
| Log Records (Page 1 of 1 Filter matched 5 of 5 records) | | | | | |
| Creation time | Severity | Message | Priority | Situation Type | |
| Apr 27, 2007 7:12:49.261000 PM | 19 | RPXE9999 write the response string from CO ... | 0 | ReportSituation | P |
| Apr 27, 2007 7:12:49.261000 PM | 19 | RPXE9999 showCReqMenu(36823030,12 | 0 | ReportSituation | P |
| Apr 27, 2007 7:12:49.261000 PM | 19 | RPXE9999 36823030 | 0 | ReportSituation | P |
| Apr 27, 2007 7:10:42.347000 PM | 49 | RPXE0035W Finish read exception for action... | 0 | ReportSituation | P |
| Apr 27, 2007 7:08:42.338000 PM | 49 | RPXE0035W Finish read exception for action... | 0 | ReportSituation | P |

Figure 13-12 Problem determination log entries

To view only those log entries written by RPT custom code, use perspective filtering to select only those entries containing the string RPXE9999.

Test log

The second type of log, called the *test log*, is supported by the ITestLogManager interface.

This interface is used to report test verdicts based on checks or customized validations performed by custom code. Five logging levels are supported from least to most detailed: Schedule, Primary Test Actions, Secondary Test Actions, Action Details, and All.

The problem determination log should be used if other information is reported during test execution. And unlike the problem determination log, the results reported by the test log can be viewed directly within the RPT Test perspective.

It is a best practice to conditionalize any information to be written to the test log with a call to the wouldReport method of the ITestLogManager object:

```
ITestLogManager testLogManager = tes.getTestLogManager();
if (testLogManager.wouldReport(ALL)) {
    testLogManager.reportVerdict ("CustomCode1: User is not yet
        logged in, a test element ordering error",
        VerdictEvent.VERDICT_FAIL,
        VerdictEvent.REASON_SEE_DESCRIPTION);
}
```

Where tes is an instance of ITestExecutionServices.

Note: To use the VerdictEvent class, be sure to import the org.eclipse.hyades.test.common.event.VerdictEvent package.

The level of detail of test log events reported by custom code to the RPT workbench is controlled by the What To Log RPT schedule setting (Figure 13-13).

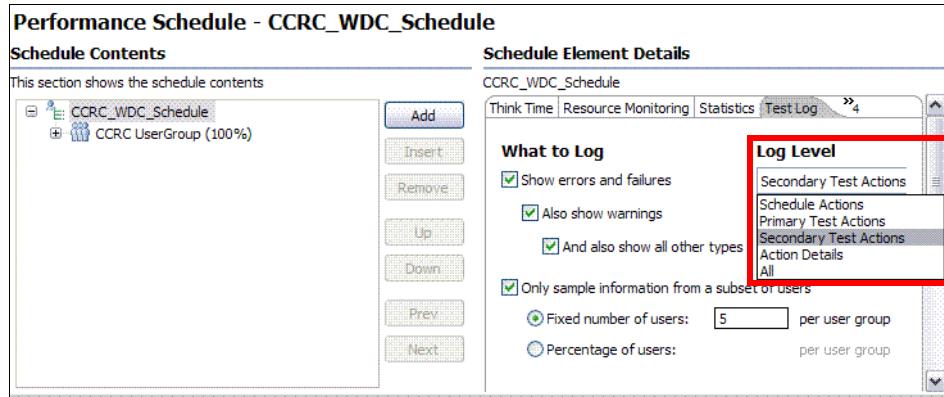


Figure 13-13 Selecting the Test Log Logging Level

13.4.2 Data areas

RPT custom code support provides data areas as a convenient mechanism to persist information for varying time frames. This mechanism can be used to save information for later use and to pass information from one test to another, or retrieve information about the current circumstances regarding a test or virtual user.

This support is implemented via the `IDataArea` interface, and this interface defines functionality for storing and accessing elements in data areas. The elements of a data area are similar to program variables and are scoped to the owning data area (or container).

There are three kinds of data areas:

- ▶ `ITestInfo`—This kind of data area persists for the length of the RPT performance test and each performance test has its own unique instance. Since a virtual user can execute several performance tests during its session, this kind of data area spans the shortest time frame.
- ▶ `IVirtualUserInfo`—This kind of data area persists for the virtual user's session across the multiple tests executed by that virtual user. Each virtual user has its own unique instance.
- ▶ `IEngineInfo`—This kind of data area persists for the duration of the RPT schedule. Engine data is visible to and shared among all virtual user sessions on that test agent. Any test running on any virtual user (on that test agent) can create, access, or modify the values set in the engine data area.

It is important to understand each kind of data area and when to use it. All three kinds of data areas provide information that can be read or retrieved. However, of these three kinds of data areas, only the `VirtualUserInfo` data area allows a user to store or write information. As a result, the `VirtualUserInfo` data area is most often used within RPT custom code because it provides a mechanism to persist information and pass it from one performance test to another.

The `ITestInfo` interface provides read-only context information about the test that is currently executing. Some commonly used methods of the interface are:

- ▶ `getName`—Provides the name of the performance test currently running.
- ▶ `getTestLogLevel`—Provides the current test log level for the currently running performance test.
- ▶ `getPDLogLevel`—Provides the current problem determination log level for the currently running performance test.

The following code fragment provides a simple example of how to use this data area:

```
ITestInfo testInfo = (ITestInfo) tes.findDataArea(IDataArea.TEST);
String perfTestName = (String) testInfo.getName();
```

The `IVirtualUserInfo` interface provides information about the current virtual user, as well as an area where information can be stored and later retrieved. Some commonly used methods of the interface are:

- ▶ `get`—Retrieves (reads) the value for an element from the current data area.
- ▶ `getUserName`—Returns current virtual user's name.
- ▶ `getUID`—Returns the unique ID associated with the current virtual user.
- ▶ `getRandom`—Returns the current virtual user's random generator.
- ▶ `put`—Adds (writes) a value for an element to the current data area.

The following code fragment provides a simple example of how to use this data area:

```
IVirtualUserInfo vuInfo = (IVirtualUserInfo)
                           tes.findDataArea (IDataArea.VIRTUALUSER);
ArrayList folderList = new ArrayList();
// populate folderList ...
vuInfo.put("myTestFolderList", folderList);
```

The `IEngineInfo` interface provides information about the performance testing execution engine. Some commonly used methods of the interface are:

- ▶ `getActiveUsers`—Returns the number of active virtual users.
- ▶ `getCompletedUsers`—Returns the number of completed users

- ▶ `getHostName`—Returns the host name on which the engine is running.
- ▶ `getDeploymentDirectory`—Returns the deployment directory in which RPT has deployed all its test assets.

The following code fragment provides a simple example of how to use this data area:

```
IEngineInfo engInfo = (IEngineInfo) tes
    .findDataArea(IDataArea.ENGINE);
int activeUsers = engInfo.getActiveUsers();
```

Through selective use of these data areas, RPT custom code can both collect needed information about the environment in which the test code is running, while also saving and later retrieving data for later use.

13.4.3 Loop control

The `ITestExecutionServices` interface provides access to another interface that can be used to control the execution flow of RPT custom code. A second interface, `ILoopControl1`, provides a mechanism that can be used within RPT custom code to break out of a loop or skip to the next loop iteration for a loop in an RPT schedule or within a performance test.

The following `ILoopControl1` methods are available:

- ▶ `breakLoop`—Breaks out of the innermost loop as soon as the current child of the loop finishes execution.
- ▶ `continueLoop`—Causes the innermost loop to skip the remainder of the current iteration after the current child of the loop finishes execution.
- ▶ `getIterationCount`—Returns the iteration for the current innermost loop.

For a specific example using `ILoopControl1`, see [Test duration control example](#).

13.4.4 ITransaction

The `ITestExecutionServices` interface provides access to another interface that can be used to define and control transactions recorded by RPT. A third interface, `ITransaction`, provides a mechanism that can be used to define, start, and stop custom RPT transactions.

This interface provides a level of flexibility not available to transactions defined in record-and-playback tests, where the transaction is defined at test edit-time. Use of the `ITransaction` interface allows transaction definition and control at test run-time.

The important ITransaction methods are:

- ▶ start—Begin the process of timing the transaction.
- ▶ stop—Ends the process of timing the transaction.

The following code fragment provides a simple example of how to use transactions:

```
ITransaction transaction = tes.getTransaction("myTransaction");
transaction.start();
// execute the transaction ...
transaction.stop();
```

Although the prior example shows how to start and stop a custom code transaction, the following example demonstrates some of flexibility and power this interface provides.

Most RPT test runs are composed of three phases:

- ▶ A phase where virtual users are gradually introduced to the system under test using a staggered start (refer to “Starting users at different times” on page 168 in 6.4.3, “Controlling user behavior”). For purposes of this example, this phase will be called the *ramp-up period*.
- ▶ A subsequent phase where all virtual users are active on the system under test. For purposes of this example, this phase will be called the *full-load period*.
- ▶ A final phase where simulated users gradually leave the system under test in the same fashion they became active during the first phase. That is, the first user that became active is the first user that leaves the system under test or exits. For purposes of this example, this phase will be called the *ramp-down period*.

These periods are shown in Figure 13-14.

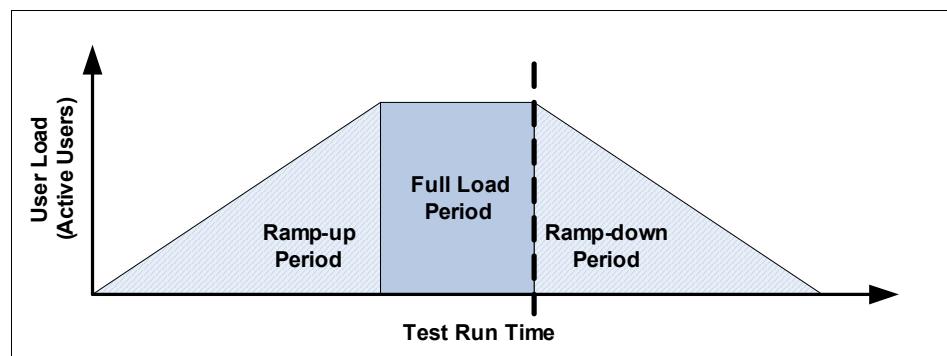


Figure 13-14 Test run user load profile

For convenience, transaction response time data is often sorted by name. This practice places an additional importance on the form and format of transaction names. By formatting transactions using the following technique, the data collected during the full-load period of a test run can be separated from the remaining data, allowing easier analysis of the system under test under full-load conditions.

The aforementioned technique is to include a transaction name substring that is based on the number of currently active simulated users. The following code fragment shows an example of how to implement this technique:

```
public String getTransactionName(ITestExecutionServices tes) {  
    IEngineInfo engInfo = (IEngineInfo) tes  
        .findDataArea(IDataArea.ENGINE).get(IEngineInfo.KEY);  
    String tranNamePrefix = new String();  
  
    // check number of active VUs and form transaction name  
    // prefix accordingly  
    // the 'magic' value "73" could be replaced with a reference  
    // to a IVirtualUserInfo data area value  
    if (engInfo.getActiveUsers() < 73)  
        { tranNamePrefix = "ramp_"+CCRC_"; }  
    if (engInfo.getActiveUsers() >= 73)  
        { tranNamePrefix = "load_"+CCRC_"; }  
    return tranNamePrefix;  
}
```

This method creates a prefix (either ramp_ or load_) that can then be pre-pended to a transaction name string depending on the then current number of active users. This will effectively separate transactions executed during the full-load period of the test run from transactions executed during the ramp-up and ramp-down periods. When the test run transaction data is analyzed and sorted after the test run, the transactions executed during the full-load period are grouped together, while the transactions executed during the other two phases are grouped together.

The aforementioned method would be used by calling it and then using the returned string as a prefix to the final transaction name string. The following code fragment shows this usage:

```
/* execute a CleardiffPred transaction */  
// (prefix) (transaction base name)  
transactionName = getTransactionName(tes)+ "CleardiffPred" ;  
ITransaction transaction = tes.getTransaction(transactionName);  
transaction.start();  
.....  
transaction.stop();
```

Important: Avoid unintentionally creating large numbers (1,000s) of differently named transactions. Each transaction causes the creation of minimum, maximum, average, and standard deviation counters for that transaction. An unintended side-effect of the creation of large numbers of differently named transactions is the transfer of a very large amount of test data back to the RPT workbench, possibly causing it to crash or hang.

13.4.5 Using other Java APIs

During the course of developing Java test code to be executed through RPT custom code support, the need can arise to use Java code from another provider, or provide Java test code developed for one RPT project for use in another project.

There are two approaches that can be used to add a Java code or an API to an RPT project:

- ▶ Import Java code as a JAR file
- ▶ Reuse through export/import

Import Java code as a JAR file

The first method is to import the Java code as a Java archive (JAR file).

To do so, right-click the project and select *Properties* → *Java Build Path* → *Libraries*, and then click *Add External JARs* (Figure 13-15).

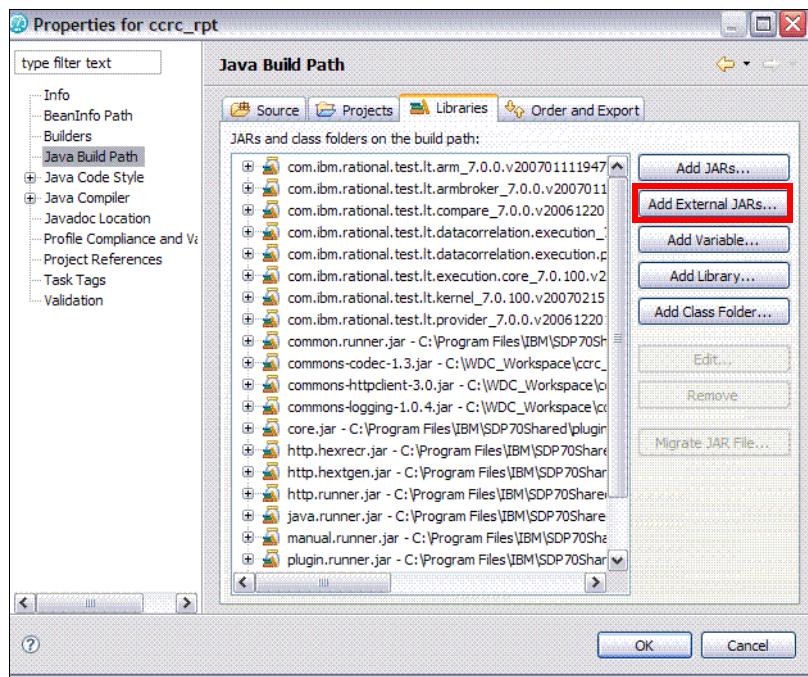


Figure 13-15 Adding an external JAR to an RPT project

In the JAR Selection dialog, navigate to the desired JAR file, click *Open*, and the JAR file is imported into the RPT project (Figure 13-16).

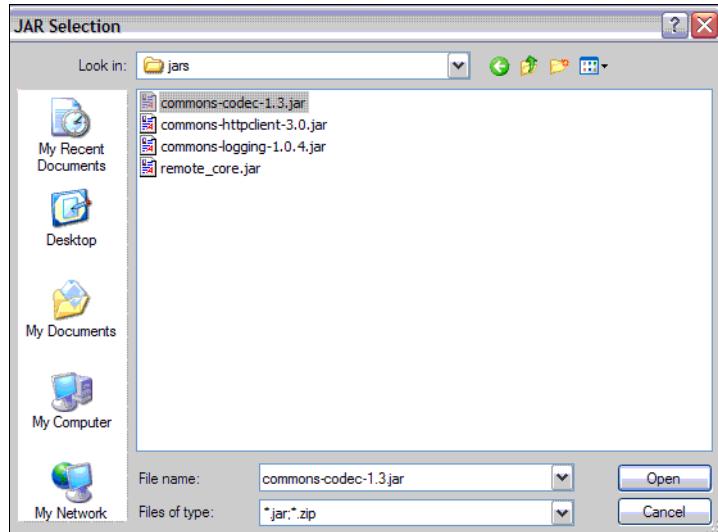


Figure 13-16 Selecting a JAR to add

After the JAR file has been imported, be sure to clean and recompile the RPT project code:

- ▶ Select the project in the Test Navigator and *Project* → *Clean* (Figure 13-17).
- ▶ In the Clean dialog, select the desired RPT project and click *OK*.

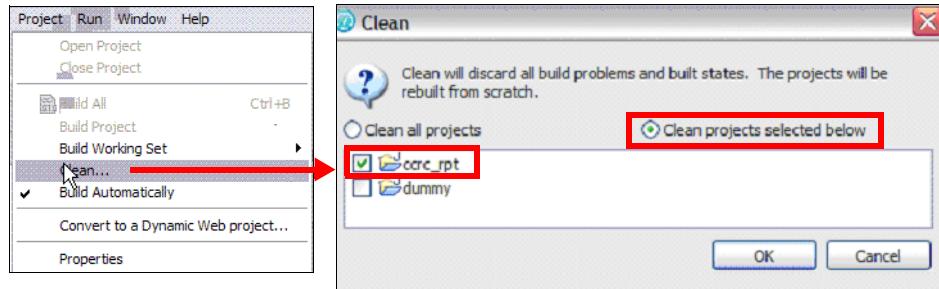


Figure 13-17 Cleaning an RPT project

At this point, the Java classes within the imported JAR file can now be accessed within RPT custom code for the project.

Reuse through export/import

A second method of Java code reuse focuses on providing the Java code developed for one RPT project to another project. This method makes use of the export/import support provided by RPT itself, where the Java code is first exported to a Zip archive file, and then that archive is imported into the other RPT project.

This procedure is as follows:

- ▶ In the Test Navigator, right-click on the desired RPT project and select *Export* (Figure 13-18).
- ▶ In the Select dialog, select *General* → *Archive File*, and click *Next*.

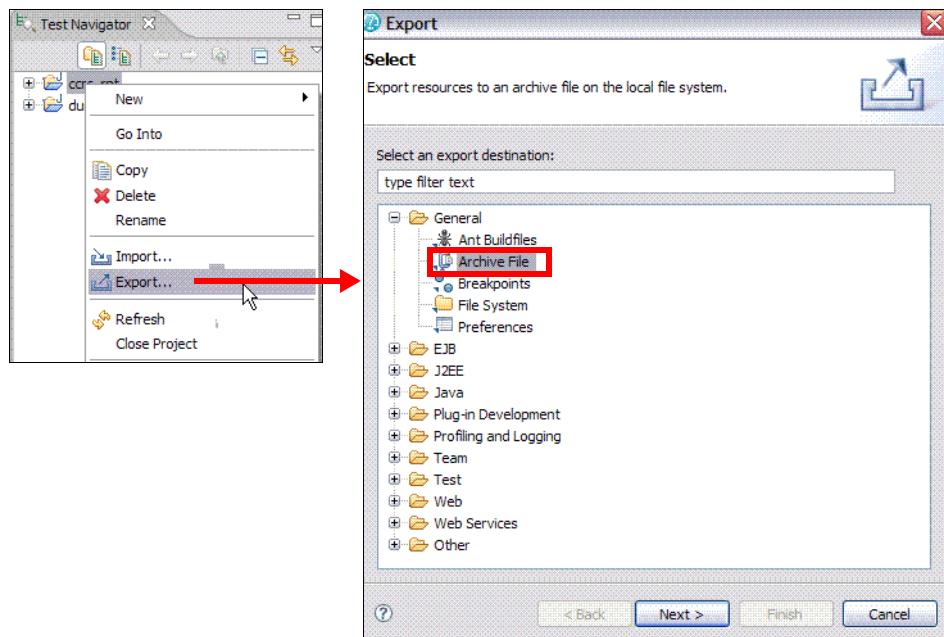


Figure 13-18 Launching an export to an archive file

- ▶ In the Archive File dialog, select the RPT project and project assets to be exported.

Note: Be sure to select .classpath, .project, and dd.testsuite.

- ▶ If desired, browse to a location where the archive is created or simply enter a path and file for the archive file. Select any other options and click *Finish* (Figure 13-19).

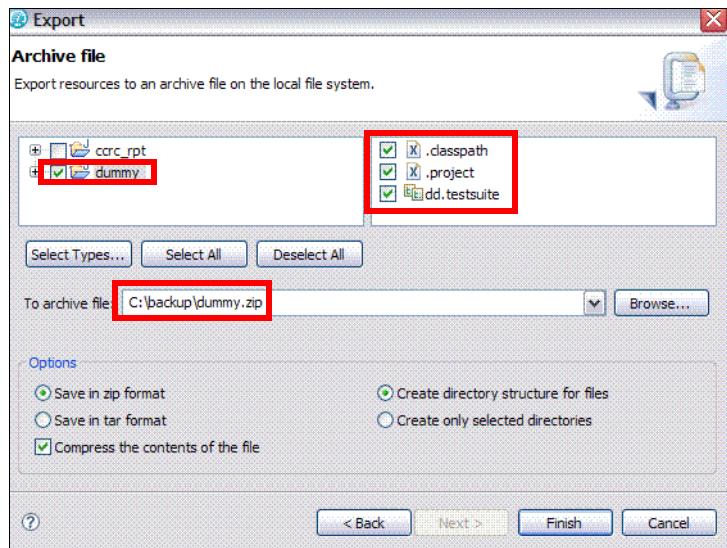


Figure 13-19 Selecting an RPT project and project elements to export

When the project has been exported, a different RPT project can be opened and the export archive can then be imported into that project by following a similar import process.

13.5 Using datapools

Frequently, modifications are made to an RPT performance test to vary its interactions with the AUT so that the same user responses are not repeated over and over. RPT provides a useful mechanism, the datapool (DP), which can be used to provide any number of user responses suitable for the AUT, in the abstracted form of a number of rows, each of which is comprised of a number of fields.

Use of datapools and custom code can be combined. When this is done, the datapool is usually used for one of two purposes:

- ▶ To provide a variety of user response data (as previously described) for use within the custom code
- ▶ To provide configuration or initialization information to the custom code.

As described in 4.4.5, “Adding datapools to tests” on page 70, RPT datapools are associated with performance tests. Similarly, custom code is also associated with a performance test. As a result, any datapool associated with a performance test can also be used by custom code associated with the same performance test.

The following example illustrates how to associate a datapool (CCRC_serverPool) to a performance test (CCRC_SAMPLELOGIN) and then makes the information provided in the datapool available to custom code associated with the test.

For brevity, a custom code element has already been added to the performance test. For a description of how to insert a custom code element into a performance test, see 13.3.1, “Adding custom code” on page 432.

The first step is to associate (add) the datapool to the test:

- ▶ Select the test and click *Add Datapool* in the Test Element Details pane (Figure 13-20).

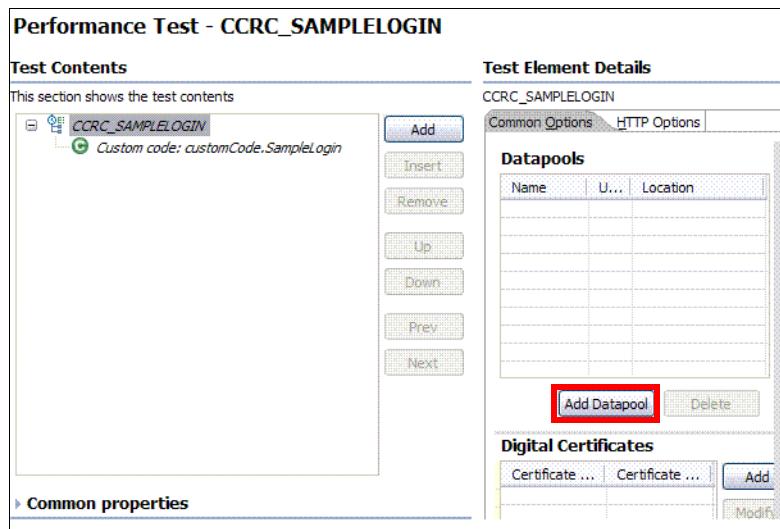


Figure 13-20 Adding a datapool to a performance test

- ▶ In the subsequent dialog, select the desired datapool and Open mode (*Segmented*, *Shared*, or *Private*) to be used when accessing the datapool, and click *Select* (Figure 13-21).

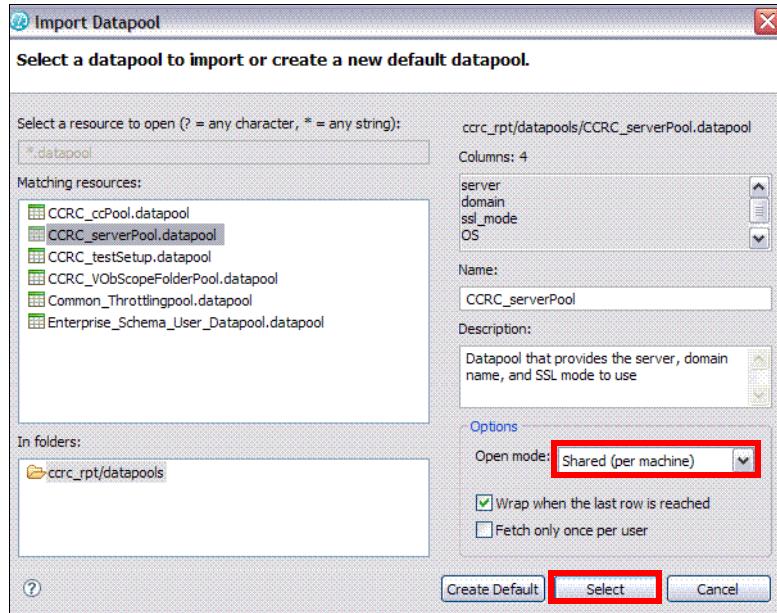


Figure 13-21 Selecting a datapool and its mode

- The datapool is associated with the performance test and displayed in the Test Element Details pane (Figure 13-22).

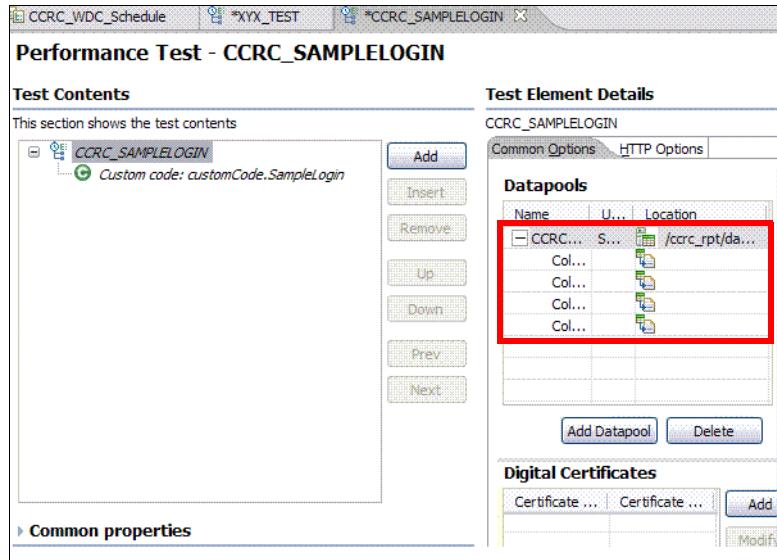


Figure 13-22 A performance test with an associated datapool

When the performance test has an associated datapool, the custom code element that is associated with the test can access and use the information provided by the datapool.

The custom code does not have to use all the fields available in the datapool, but it can select which fields to use.

To pass selected information from the datapool to the custom code:

- ▶ Select the custom code element in the Test Contents window and click *Add* (Figure 13-23).

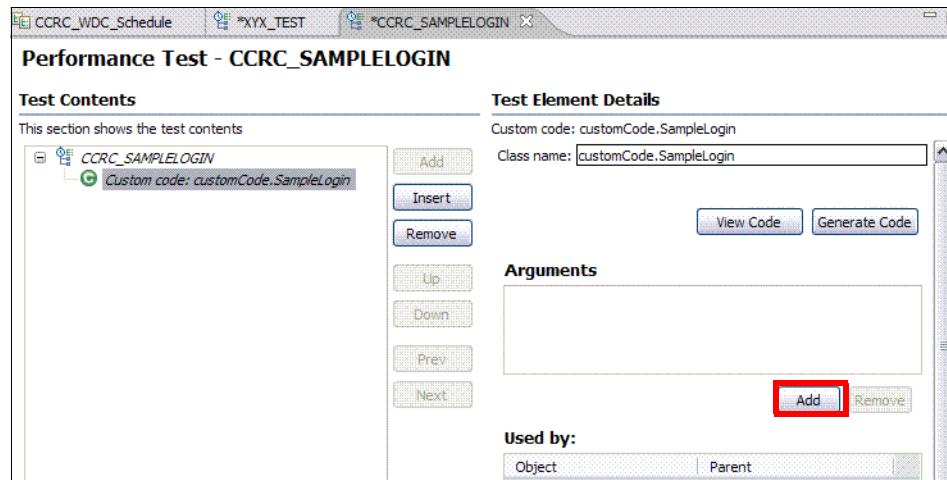


Figure 13-23 Adding datapool data to a custom code element

- ▶ In the Select Arguments dialog, select the datapool fields desired, and click *OK* (Figure 13-24).

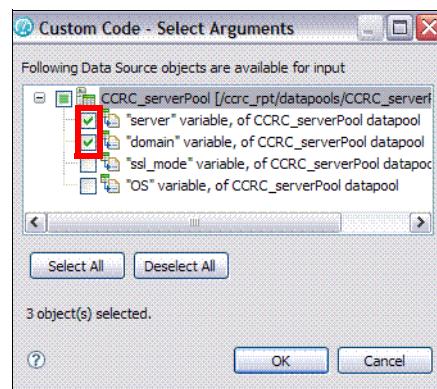


Figure 13-24 Selecting datapool fields for custom code use

As stated in 13.3.1, “Adding custom code” on page 432, a custom code element is required to accept two arguments. One of these is an instance of ITestExecutionServices, and the other is a String array.

The String array is used to pass values (as arguments) from the datapool to the custom code. Each field from the datapool that was selected is one of the String array values. These values can be easily indexed and used for further processing within the corresponding custom code exec method.

The following example illustrates the use of datapool arguments with custom code. The example assumes two datapool fields, username and password, have been selected to be passed as arguments to the custom code:

```
args[0] = username  
args[1] = password
```

The following code fragment is used for this example:

```
public class myDataValidation implements ICustomCode2 {  
  
    public String exec(ITestExecutionServices tes, String[] args) {  
  
        IDataArea dataArea = tes.findDataArea(IDataArea.VIRTUALUSER);  
  
        // Write the username to the VU data area  
        dataArea.put("Test_username", args[0]);  
  
        // Write the password to the VU data area  
        dataArea.put("Test_password", args[1]);  
  
        // further processing ...  
    }  
}
```

Using the same technique, initialization or configuration information for a custom code test can be provided by a datapool. Examples of such information include: ClearCase view storage directory paths, ClearQuest® schema names, server host names, privileged user account names, and so forth.

When associating the datapool to the performance test, use the *Private* open mode so that each performance test receives the same information.

13.6 Extending tests with custom code

Thus far, this chapter has provided an overview of some of the important aspects of RPT custom code support and its use.

In this section we now turn to the topic of how to put this support to practical use.

13.6.1 Passing data to custom code

One practical application of custom code is custom test data validation, as shown in 13.3.2, “An example of custom code use” on page 435. To accomplish this within the context of an RPT HTTP record and playback performance test, an HTTP response must be passed to a custom code class for validation processing.

The following example shows how to pass an HTTP response to a custom code class.:

- ▶ To begin, locate the HTTP response in the performance test that has to be passed to a custom code class, and right-click on the response to create a field reference (Figure 13-25).

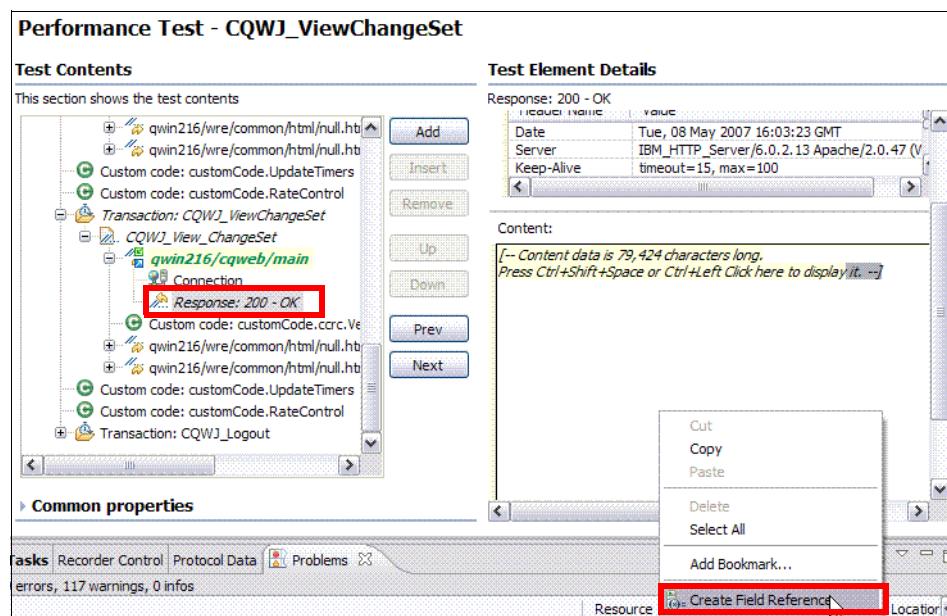


Figure 13-25 Selecting an HTTP response for field reference creation

- ▶ Next, click *Add* and select *Custom Code* to add a custom code element to the performance test after the selected HTTP response (Figure 13-26).

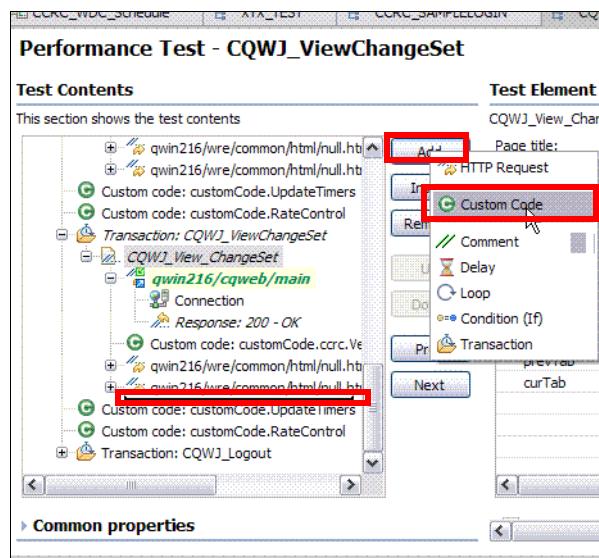


Figure 13-26 Inserting a custom code element

- ▶ Now select the newly added custom code element and click *Add* in the Arguments pane to provide the recently created field reference as an input argument (Figure 13-27).

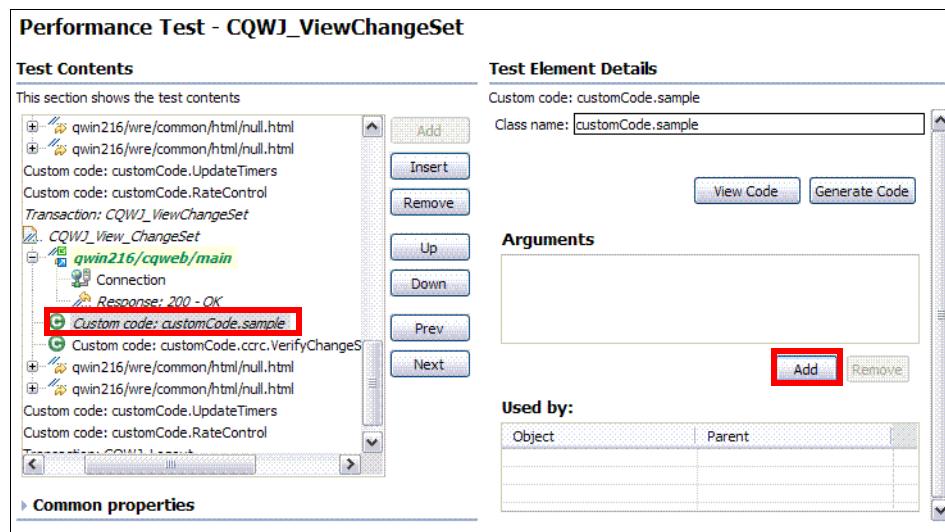


Figure 13-27 Adding an input argument to a custom code element

- In the Select Arguments dialog, select the field reference as an input argument (Figure 13-28).

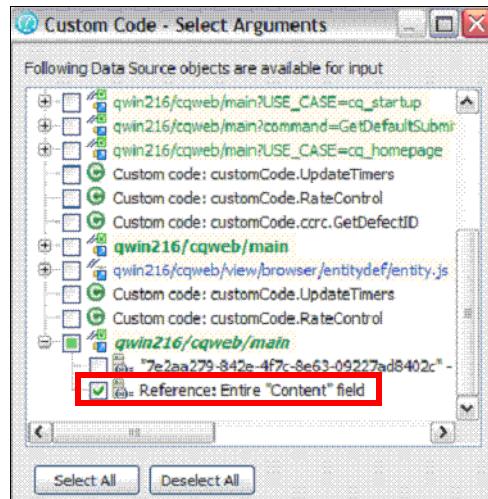


Figure 13-28 Selecting a field reference as an input argument

This makes the HTTP response available to the exec method of the custom code class for further processing through the args String input array.

When that is done, reference the passed HTTP response in the custom code as args[0] to do any validation processing inside the exec method.

13.6.2 Returning data from custom code

When the custom code class has processed the response, it might be desirable to reverse this process, and return processed data from a custom code class to a performance test, substituting the data for a previously recorded HTTP response.

To return data from a custom code class to a performance test, the String return value of the custom code class exec method is used.

Consider the following example custom code class, which retrieves a password value from the virtual user data area, and then returns this value:

```
public String exec(ITestExecutionServices tes, String[] args) {
    IPDLogManager ipdLogManager = tes.getPDLogManager();
    IDataArea dataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
    // Retrieve password value
    String pwd = (String)dataArea.get("sptCCRC_pwd");
    return pwd;
}
```

As highlighted in the example, the class returns a String value of `pwd`, which is the desired password string to be used in the performance test.

This custom code class method can be used in conjunction with the *Substitute from custom code* option within RPT to use the return value from a custom code class in a HTTP performance test.

The following example illustrates this substitution procedure. For this example, a CQWJ_ViewChangeSet performance test includes a ClearQuest Web login HTTP request, in which the user password originally recorded is to be substituted with a password provided by a custom code class:

- ▶ To begin, add the custom code element to the performance test just before the HTTP response in which the response substitution is to take place. Then select the HTTP page in the Test Contents pane, right-click on the response value in the Test Elements Details pane that is to undergo substitution, and select *Substitute From* and select the custom code class that was inserted into the test just before the response (Figure 13-29).

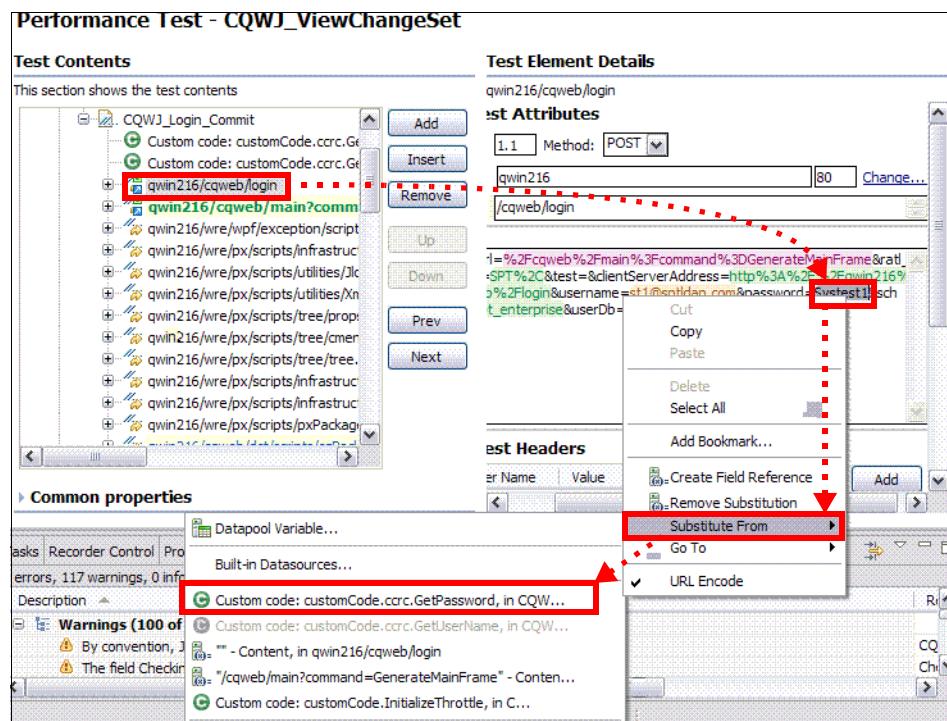


Figure 13-29 Substituting a response value with custom code

When the schedule is run again, and the performance test is executed, the response used in the HTTP page will be provided by the custom code class, not the value originally recorded.

13.6.3 Test duration control example

The example presented in this section illustrates the use of custom code to control performance test or schedule control flow.

For large-scale test runs simulating hundreds or thousands of virtual users, the ramp-up period time (that is, the time during which virtual users are introduced into the test run and become active) can often consume a considerable amount of time, especially in case where test start stagger periods are long.

Consider a 1,000 user test run with a stagger time of two minutes (120 seconds). Such a test run would have a load ramp-up time of 999 multiplied by 120 seconds, or 119,880 second (33.3 hours). If the schedule consists of performance tests that iterate for a fixed number of iterations such that some portion of those iterations are executed under a full load, the resulting test run load profile resembles a trapezoid, as shown previously in Figure 13-14 on page 446.

As is shown in the load profile in that figure, a significant amount of time is also spent in the ramp-down period. If the objective of the test run is to collect product performance data during the full load period, a large amount of time is spent in ramp-up and ramp-down periods, slowing testing progress if multiple test runs are desired.

One optimization that will improve testing efficiency is to limit the amount of time spent in the ramp periods. While the ramp-up period is necessary to avoid overwhelming the test system as the total user load is introduced into the system, the ramp-down period is not. In fact, if the ramp-down period can be reduced or eliminated, significant time savings will result.

One solution to realize this time savings is to use custom code at the end of each test schedule main loop iteration to determine if the desired test run elapsed time has reached the end of the full load period, indicated by the dashed vertical line in the Test run user load profile. The custom code class would determine the test run elapsed time thus far, and whether the full load interval was complete. If it was complete, the custom code breaks out of the main schedule loop.

The following code fragment implements this elapsed time check and breaks out of innermost loop when the desired time limit has been passed:

```
public String exec(ITestExecutionServices tes, String[] args) {
    // Get the test duration limit stored in virtual user data area
    // Get the IPDLogManager to report test execution information
    IPDLogManager ipdLogManager = tes.getPDLogManager();
    String testDuration = args[0];
    int durationMS;
    try {
        ILoopControl loopcontrol = tes.getLoopControl();
        ITime itime = tes.getTime();
        IDataArea dataArea =
            tes.findDataArea(IDataArea.VIRTUALUSER);

        // Fetch the test time limit from the VU data area
        String testDuration = (String) dataArea
            .get("Test_Time_Limit");
        Integer intDuration = new Integer(testDuration);
        // minutes*60*1000 = milliseconds
        durationMS = intDuration.intValue() * 60 * 1000;
        if (itime.timeInTest() > durationMS) {
            // break out of inner-most loop
            ipdLogManager.log(IPDLogManager.FINE,
                "Test time limit elapsed");
            loopcontrol.breakLoop();
        } else {
            ipdLogManager.log(IPDLogManager.FINE,
                "Test time still remains");
        } catch(Exception e) {
            ipdLogManager.log(IPDLogManager.SEVERE,
                "Exception in Duration control" + e.getMessage());
        }
    return null;
}
```

A custom code element, with the aforementioned implementation for its exec method, would cause each virtual user to break out of the main schedule loop once the time-in-test had surpassed a time limit specified in minutes and previously stored in the virtual user data area for each user.

13.7 Synthetic workload simulation

The primary method for RPT test automation development is AUT user interaction record and playback, where the user's interactions with the AUT are recorded at some protocol level, and then later played back to simulate one or more virtual users.

However, there are occasions where this method is not viable: Client/server applications that do not support a Web browser client, or use a protocol that RPT does not support.

In these cases, there can be another option. If the application provides a client Java API or a client API that can be called using Java, then RPT custom code support can be leveraged to provide AUT test automation.

This alternate method, herein referred to a *synthetic workload simulation*, uses RPT custom code support to execute Java test code built on top of the AUT client API. Refer to Figure 13-1 on page 431 for a high-level architectural view of this testing method.

13.7.1 Prerequisites

An important factor to consider when using a synthetic workload approach is whether the AUT client API to be used also interacts with a system's GUI desktop. RPT tests that trigger such interactions will encounter problems, typically test time-outs, as no human user will respond to the pop-up window or dialog that has been triggered.

A synthetic workload approach essentially entails the development of a *headless* client test application written in Java. In this context, "headless" means that the resulting Java test application should not interact with a system desktop in any form (that is, it should not trigger pop-up windows or dialogs, or solicit manual input from a user). These behaviors are not handled by RPT, and cause fatal problems for any test that triggers such behavior.

With the aforementioned restriction in mind, a careful review of the product client API to use is warranted. A key question to evaluate during this review is: *Can the desired test workload be generated by API calls that do not trigger desktop interaction?* If this question can be answered with yes, then the synthetic workload generation approach can be used.

13.7.2 Recommended practices

Similar to any other Java programming project, there are a number of recommended practices that will make the project easier.

These practices include those for development of well-written Java code, and should likewise be followed in any RPT custom code synthetic workload test development effort.

In addition to these, we also offer the following recommendations:

- ▶ Where possible, implement one product client transaction (or command) per RPT performance test.

Doing so will significantly benefit test artifact readability and maintainability, especially in cases where the test automation is to be developed by a team.

- ▶ Consolidate test initialization and setup so that it is performed by a single RPT test.

While in reality there can be several methods from various custom code classes that handle portions of test initialization and setup functionality, having one performance test that handles test initialization and setup will be important.

This performance test will collect no response time data. However, all subsequent performance tests, and the custom code classes they invoke will have as a pre-condition that this initialization/setup test has executed.

The remaining custom code classes which support execution of various AUT client transactions will thereby be free to focus on put() and set() calls for data area variables, significantly simplifying their implementation.

13.7.3 Application under test state management

Another important topic to consider when developing synthetic workload test automation is management of the state of the application under test (AUT) client.

Consider simulation of a Rational ClearQuest Eclipse Client (CQEC) user. Before the simulated user can perform any use case, the user must first login and authenticate using a set of user credentials to connect to a specific user database for a specific schema repository. When that has been successfully accomplished, it might be desirable to have the simulated user modify a defect record. But which record? Does the simulated user have the appropriate permissions to modify records?

A simple solution to this issue might be to have the simulated user first create a new defect record, and then save the corresponding record ID for later modification. Because the simulated user created the record, the user should have the necessary access privileges needed to modify the record, including adding an attachment to the record.

As can be seen in this example, the AUT client has an associated state (user is logged in, has created a defect, and so forth). Figure 13-30 illustrates an example of AUT client state transitions.

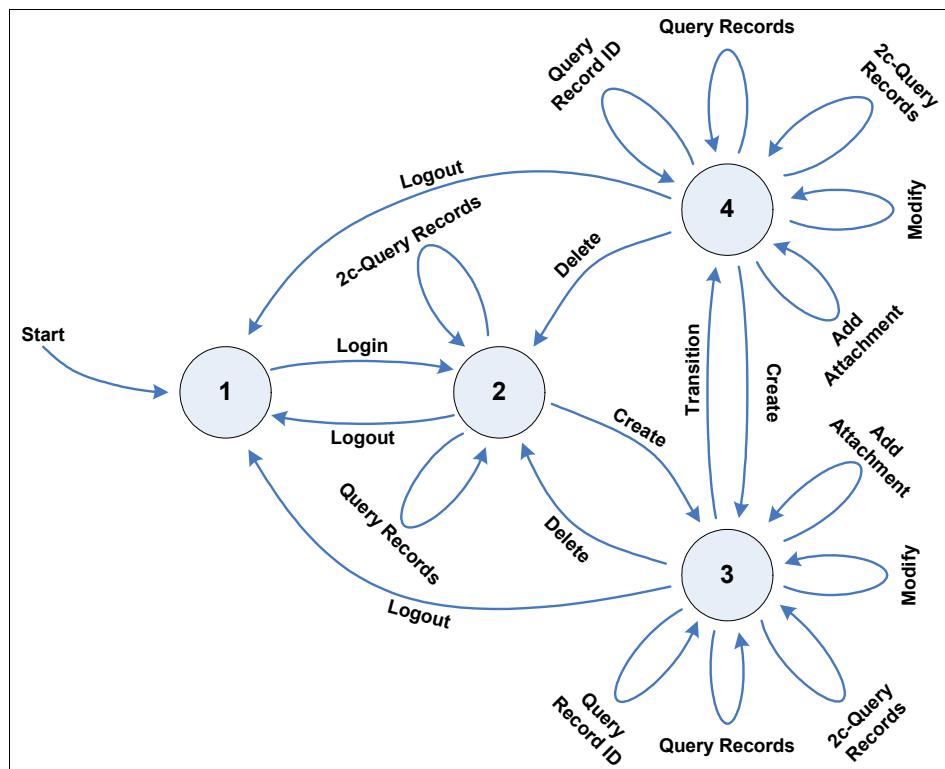


Figure 13-30 Test application state transition diagram example

In Figure 13-30, AUT client states are represented as numbered circles, while labeled directed arcs (lines with an arrow at one end) represent execution of a specific (labeled) use case. These directed arcs can cause the AUT client to enter a new state (as viewed by the test automation) or remain in the same state.

Test automation simulating a user must manage this AUT client state to operate within the restrictions imposed by the product.

A simple solution to this state management requirement is to create RPT test scripts that perform a set of user actions where the starting and ending application states are the same. In the aforementioned ClearQuest example, such a sequence might be:

- ▶ Login
- ▶ Create defect
- ▶ Run query for defect
- ▶ Modify defect
- ▶ Logout

RPT record-and-playback tests are constrained to follow this AUT client state management strategy, because support for RPT *split-script* execution is not available.

There are at least two potential issues with this approach. One issue is that this approach can force the execution of certain use cases (for example, login and logout) more often than desired. The second issue is that a significant amount of work must be done to reach a desired state, and that state does not persist beyond the test script execution period.

However, custom code-based RPT tests are not constrained to follow this same strategy. An RPT test script can invoke custom code classes in any desired sequence. This flexibility allows the definition of an AUT client *home* state other than *inactive* (the not logged-in state given the previous ClearQuest example). Recognition of this can lead to RPT schedules executing custom code-based tests to take on a recognizable structure:

```
initialization
home state setup
main loop
  selector
    use case sequence #1
    use case sequence #2
    use case sequence #3
    .....
cleanup
```

- ▶ The **initialization** phase handles any required test automation setup, such as population of information (for example, test server host name, test database name, use case execute rate) into the `IVirtualUserInfo` data area. This information is typically read from an initialization datapool (see 13.5, “Using datapools” on page 452).
- ▶ The **home state setup** phase handles execution of any required use cases to enter the desired AUT client home state. For example, login to a specific database, using the previous ClearQuest example.

- ▶ The **main loop** of the schedule iterates over execution of an RPT **selector**, which in turn executes RPT tests each of which executes a desired sequence of AUT client use cases (**use case sequences**). Each of these tests has the same starting conditions that correspond to the AUT client home state and after execution, return the AUT client to the same home state. These use case sequence can be arbitrarily complex and provide a great deal of flexibility in creating a desired workload for the AUT.
- ▶ After the main loop completes execution, the **cleanup** phase handles any use case execution to gracefully and cleanly shut down the AUT.

Figure 13-31 shows an example of this type of RPT schedule.

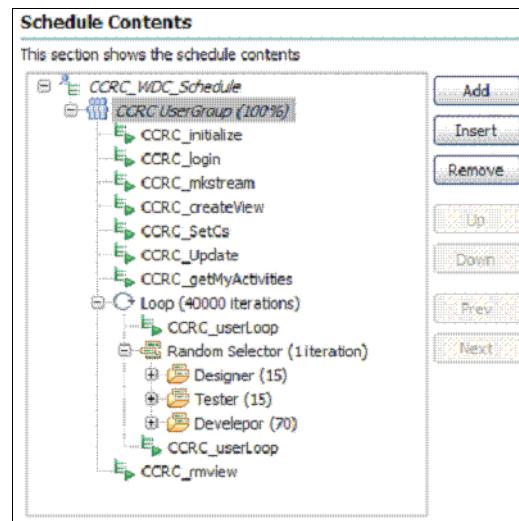


Figure 13-31 Test schedule high-level structure

In Figure 13-31:

- ▶ The CCRC_initialize script handles the initialization phase.
- ▶ A sequence of performance tests that execute the initialize, login, mkstream (make UCM stream), createView (create UCM view), SetCs (set view config spec), update (update CCRC view), and getMyActivities (get a list of my activities) implements the home state setup phase.
- ▶ The (main) loop iterates for 40,000 iterations, and invokes:
 - A CCRC_userLoop test, which further controls the behavior of loop and serves to break-out of the loop after a specified time period has elapsed (see 13.6.3, “Test duration control example” on page 461). If the time period has not yet elapsed, schedule execution proceeds.

- An RPT random selector that selects one of a set of choice folders that simulates a specific type of virtual user.
- A CCRC_userLoop test, which further controls the behavior of loop, and serves to break-out of the loop after a specified time period has elapsed (refer to 13.6.3, “Test duration control example” on page 461). If the time period has not yet elapsed, schedule execution proceeds.
- ▶ The CCRC_rmview test, which removes a previously created CCRC view and handles the cleanup phase.

This example further also illustrates how an RPT schedule can be structured in such a way as to also support simulation of different types of users for the AUT. Within each of the three first-tier random selector choice folders is a second-tier selector, the choices for which correspond to use cases for that type of user. Each of the choices for these second tier selector choices corresponds to a sequence of RPT tests that correspond to a sequence of AUT client use cases.

This additional level of detail is shown in Figure 13-32.

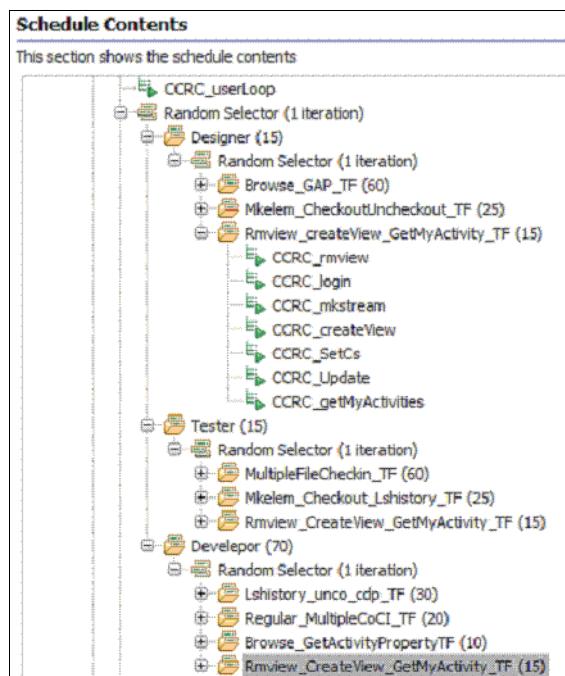


Figure 13-32 Test schedule mid-level structure

This schedule structure can be easily customized to a specific AUT workload, especially one where the overall workload is expressed as a set of user types with relative distribution percentages with each user type in turn having a unique use case distribution also expressed in percentages. The RPT schedule first-selector weights would be set based on the user-type distribution percentages, while each of the second-tier selector weights would be set based on the per-user-type use case distribution percentages.



Smart Bank: An example of dynamic workload implementation with RTP

This chapter describes how Rational Performance Tester (RPT) can be used to make a dynamic showcase, through an example of the implementation of the Smart Bank showcase workload.

We cover the following topics:

- ▶ Smart Bank: What is it?
- ▶ The demonstration proof-points: A day in the life of a bank
- ▶ Workload analysis
- ▶ Smart Bank showcase implementation
- ▶ RPT V7 architecture for workload injection
- ▶ Conclusion

14.1 Smart Bank: What is it?

The official name of the project is *Building a Smart Bank Operating Environment*.

For ease of implementation, we use the *Smart Bank showcase*:

The Smart Bank showcase was created in ATS-PSSC Montpellier, France as a vehicle to show Retail Banking customers the value of IBM infrastructure in addressing their key business issues. As such it was a project whose scope and target audience were world-wide, directed by Financial Services Sector (FSS) and Systems Technology Group (STG).

Objectives

The objectives of the Smart Bank showcase are to:

- ▶ Demonstrate the value of IBM infrastructure in the retail banking context.
- ▶ Provide a tool to engage the customer and interest him in further projects and sales with IBM hardware, software and services.
- ▶ Focus on the business value and develop proof-points to show how the infrastructure can help deliver key banking projects.
- ▶ Run the demonstration of the proof-points at operational volumes typical of a European sized bank:
 - 6 million customers
 - 12 million accounts
 - Average daily online transaction throughput of about 300 transactions/second rising to a peak of up to 1200 transactions/second
 - Constant background batch workload setup as the lowest priority workload during the demonstration
- ▶ Provide a platform for internal IBM collateral, testing of concepts and documentation:
 - We participated in the z9™-109 Early Support Programme (ESP™) project testing the new machine with near operational volumes and utilization.
 - This Redbooks publication is the first piece of external collateral to be created from the project.
 - We provided an analytical database developed in Montpellier based on the Information FrameWork's (IFW) Banking Data Warehouse (BDW) to other IBM teams.

Assets

We were able to leverage the resulting database from an Independent Solution Vendor (ISV) benchmark conducted in Montpellier (Fidelity). This included the agreement with the ISV to use their application software to represent the core system in our retail bank launched the project.

As the project grew to cover new business proof-point, we engaged other ISVs to provide the application functionality to highlight our infrastructure.

Here we list the ISVs and IBM assets engaged in the Smart Bank showcase:

- ▶ Fidelity Corebank v4.2—Real-time retail banking application based on a physical implementation of the Financial Services Data Model (FSDM), which forms the foundation of the Information FrameWork (IFW) model from IBM Software group (SWG) in Dublin.
- ▶ Fair Isaac TRIAD v8.0—Risk calculation engine
- ▶ Siebel Business Analytics v7.7.1—Business Intelligence application
- ▶ Stonesoft Stonegate v2.2.7—Linux firewall security software
- ▶ IBM Banking Data Warehouse (BDW) from IFW, built on the FSDM data model
- ▶ ACI Worldwide Ltd, BASE24-es v.6.2—Payments Framework. Refer to the Redbooks publication, *Guide to Using ACI Worldwide's BASE24-es on z/OS*, SG24-7268
- ▶ IBM ATS-PSSC, Montpellier resources, experts, hardware and software

Note: To have more information about the architecture of the Smart Bank showcase, refer to the Redbooks publication, *Infrastructure Solutions: Building a Smart Bank Operation Environment*, SG24-7113.

14.2 The demonstration proof-points: A day in the life of a bank

For the demonstration, we took these strategies and tried to tackle them in a logical order as we created our environment, each time trying to address one of the proof-points for retail banking.

In this section we cover the proof-points one by one in roughly in the order in which we addressed them.

The first thing we had to do was to integrate our modern core retail banking system provided by Fidelity to our channels to allow our customers to transact with us.

Multi-channel transformation (MCT)

This proof-point verifies our ability to:

- ▶ Reuse business logic across more than one delivery channel and open new delivery channels or enterprise product offerings rapidly.
- ▶ Manage these channels centrally and collect metrics in real-time, that could be used for billing lines-of-business based on usage.
- ▶ Control the Quality of Service and user experience across these channels.
- ▶ Provide a resilient, secure, scalable and highly available business service for these delivery channels.

Branch transformation

This proof-point addresses the branch transformation:

- ▶ For the branch channel we adopted a centralized model, where the branch servers are physically located on centralized servers to help simplify the environment and reduce costs.
- ▶ It focuses on horizontally scalable applications deployed in a distributed environment (perhaps a regional hub) or in a central server environment.
- ▶ It also inherits the points from the multi-channel transformation point and shows end to end integration across heterogeneous platforms.

Having achieved the flexibility afforded by our multi-channel architecture, we then started to tackle the *new programming model* and service-oriented architecture (SOA). The key driver for this proof-point was rapid development (quick time to market) of new business products.

Rapid product development (quicker time to market with SOA)

This proof-point shows that we are able to:

- ▶ Leverage the core system functionality by exposing it as a service to be consumed by a business process
- ▶ Build new business processes based on existing core system functionality and new functionality created in the new programming model (J2EE).

This leads into the discussion around core systems transformation.

Core systems transformation

This proof-point shows how we can:

- ▶ Identify the core competencies and functions encapsulated into our core systems and then leveraged through a service-oriented architecture (SOA).
- ▶ Identify back-office areas that can be optimized for more efficient operations. Having access to the mission critical core system functionality through SOA, it is then possible using the new programming model of J2EE, open standards, and processes to optimize these functions and add new value by:
 - Offering greater integration for cross-selling
 - Enabling transformation of the core operation
 - Providing system flexibility to better deal with new products and regulatory compliance.

Customer insight (business intelligence)

This proof-point describes how we can:

- ▶ Create customer segmentation using an analytical banking data warehouse. This identifies key customer groups to whom we will offer a new product.
- ▶ Create an architecture to update in near real-time the analytical database to allow the business to make more informed decisions on up-to-date information.

Regulatory compliance

This proof-point meets the following requirements:

- ▶ Offers regulatory reporting from the risk framework created by leveraging the analytical database created for customer insight.
- ▶ Shows how an enterprise can meet some of the demands of BASEL II through the use of a central banking data warehouse (analytical database).

Business continuity—Operation risk and business resiliency

This proof-point is designed to:

- ▶ Show how the enterprise can react to unplanned outages maintaining data integrity, consistency and security.
- ▶ Create a highly available environment with hardening of platforms, redundancy and recovery.
- ▶ Allow for scalability both horizontal and vertical, and plan for prioritizing workloads and systems.

Infrastructure simplification

This proof-point encompasses the following concepts:

- ▶ Infrastructure simplification is an underlying message behind the whole demonstration, with the monitoring and management of a system that can use virtualization and autonomic technologies.
- ▶ This, together with the systems management, is the proof-point that this document helps to address, and then provides the mechanism through which we demonstrate the other proof-points.

We do not pretend to deliver the ultimate solution to all these problems, but to show how the IBM strategies and infrastructure can be used to address them with an appropriate use of technologies and a set of example ISVs who perform certain business functions.

Online workload

The standard online workload was derived from the research into actual banks workloads and was set up to represent a typical day's online activity (Figure 14-1).

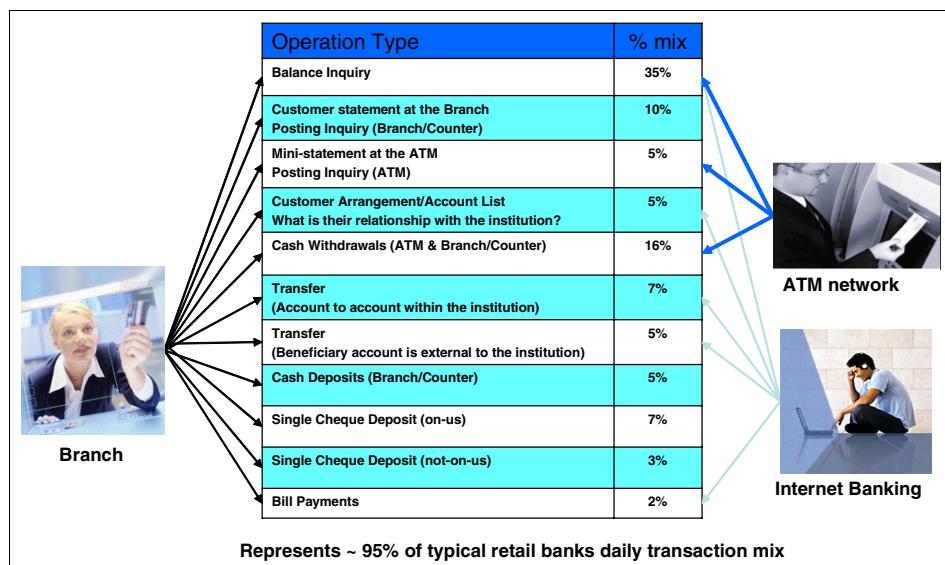


Figure 14-1 Online workload

Within this transaction mix we cover straightforward inquiries, to mini-statements showing posting activity, to all major types of online financial transactions.

This online workload can be simulated using a performance schedule (refer to Figure 14-7 on page 486).

A day in the life of the bank

We use a story to drive the demonstration and to tackle the proof-points clearly. This story involves one day's activity and starts off at 06h00 in the morning¹ as our customers are either returning home from the night shift, leaving local nightclubs, or starting the day.

Table 14-1 shows at a high level of detail the events that we typically run through for the Smart Bank showcase.

Table 14-1 The story board for the day in the life of a bank

| Time ^a | Event | Description |
|-------------------|---|---|
| 06:00 | Setting the scene. Early morning at the bank | <p>Here we show low channel traffic coming from our ATMs and Internet channels as our customers withdraw cash for the day or access our internet bank.</p> <p><i>(We show the injection mechanism, perform a business transaction, or two from our Internet bank, and emphasize that this is a real workload on a live operational system.)</i></p> |
| 09:00 | Branches open. Workload increasing | <p>A new channel comes on-line and our customers start transacting via our branches.</p> <p><i>(Here we introduce our enterprise management - Tivoli Enterprise Portal - views and show the branch channel arriving from a workload point of view and highlight the qualities of service (QoS) that we can measure across our customer facing channels.)</i></p> |
| 10:00 | Business intelligence. Customer segmentation | <p>The bank's analysts, through the use of business intelligence software, have detected a trend with our mortgage portfolio and continue to identify a customer segment and a promotional opportunity.</p> <p><i>(Now we launch into the customer profitability, revenue growth, customer segmentation proof-point, using Siebel Business Analytics.)</i></p> |
| 10:30 | Near real time business analytics data | <p>How have we created the architecture for our analytical database? and achieved a near real-time update of our analytical database?</p> <p><i>(Show the near-real time analytics update speeds through simple SQL queries from Siebel Business Analytics to BDW and describe the architecture.)</i></p> |

¹ We recognize that many banks operate across different time zones and that this can even-out or blur the peaks and troughs of a daily workload. For reasons of simplicity and of showing a more dramatic dynamic demonstration we have kept the demonstration to one time-zone for the moment.

| Time ^a | Event | Description |
|-------------------|--|---|
| 10:45 | Leverage the new programming model | The bank wants to rapidly create new business products to offer to their customers. We leverage the capabilities presented with service oriented architecture (SOA) to do this and also leverage our existing mission critical applications to do so. <i>(Show SOA workload running and describe architectural components and techniques.)</i> |
| 11:00 | Introduce business performance dashboards | A new capability made possible by service oriented architecture - a dashboard for business performance to measure see how well our channels are performing. <i>(Show the business dashboard with near-real time business metrics and describe how we achieved rapid application build.)</i> |
| 12:00 | Managing lunch time volumes | Lunch time workload increases. <i>(Show how workload management can automatically maintain QoS and Service level agreements based on business priorities.)</i> |
| 13:00 | New promotion launch. Promotion peaks internet traffic | Our promotion is working better than expected. On/off capacity on demand. <i>(Move to Hardware Management Console to initiate an OOCoD - on/off capacity on demand request. Show the effect of this request on the workload via TEP. Highlight virtualization capabilities of the platform.)</i> |
| 14:00 | Business continuity | Now we have some problems. We discuss the concepts of business continuity and from an IT point of view we have achieved Recovery, Redundancy and Hardening. <i>(Through TEP, drive the scenarios of planned outage: redundancy, unplanned disk outage; recovery and unplanned server within clustering technology; hardening.)</i> |
| 15:00 | Credit (Basel II) and operational risk | Having discussed some of the ways to counter operational risk, we now turn to credit risk reporting, another aspect of Basel II compliance. <i>(Return to Siebel Business Analytics to reuse our business intelligence framework for regulatory compliance reporting with Credit Risk.)</i> |
| 17:00 | Branches close. Transforming a branch infrastructure | Branch consolidation and the ability to have platform independence. Choose the platform dependent on quality of service. Simplification of the infrastructure. <i>(Discuss the concepts of branch transformation and how we have simplified the environment and saved costs.)</i> |

a. The times are sometimes contrived for the demonstration. The business event does not have to be restricted to this time.

Figure 14-2 shows the consolidated workload. Note that the start time of each channel and that the number of hits of each channel is not the same according to the hour of the day.

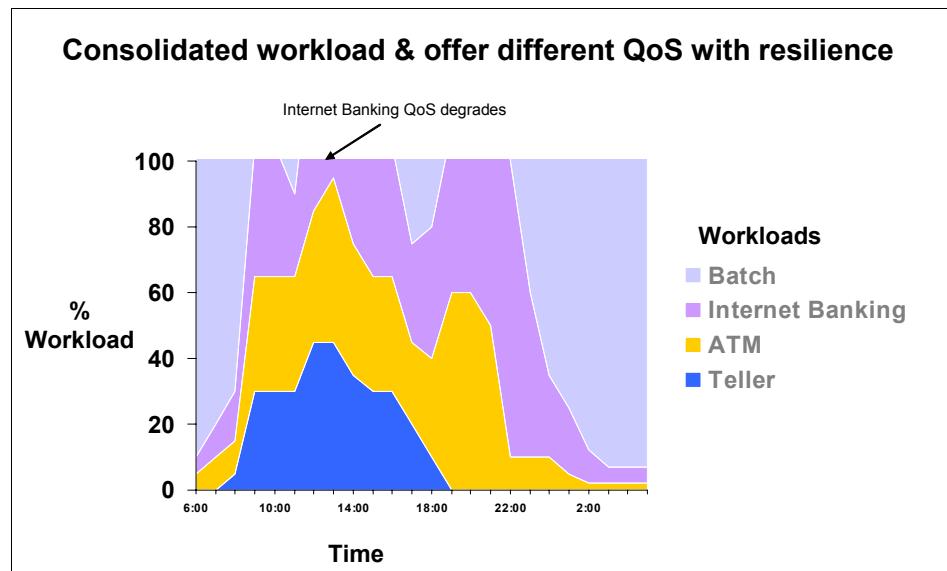


Figure 14-2 Workload chart

14.3 Workload analysis

For reasons of simplicity, we did not simulate the one day totality, but are focused at the most significant time.

For our example, we use Table 14-2 as a workload reference to simulate.

Table 14-2 Sample of workload

| Step | Time | Description (which channels) | Back-end Transactions | Hits/Sec |
|------|-------|--|-----------------------|----------|
| 1 | 04:00 | No Teller, no Process | 30 | 12 |
| 2 | 06:00 | No Teller, no Process, increase activity | 100 | 40 |
| 3 | 07:00 | Start Teller, no Process | 114 | 50 |
| 4 | 10:30 | Start Process | 145 | 70 |
| 5 | 10:30 | All channels | 200 | 90 |

| Step | Time | Description (which channels) | Back-end Transactions | Hits/Sec |
|------|--------------|--|-----------------------|----------|
| 6 | 11:00 | Increase all, more for teller and process | 320 | 135 |
| 7 | 11:30 | Increase all workloads | 340 | 165 |
| 8 | 12:00 | Increase all workloads | 740 | 360 |
| 9 | 12:00 | After capacity on demand | 1000 | 430 |
| 10 | 13:00 | Same workload as step 6, but with more CPU on z9 | 500 | 240 |

- ▶ A back-end transaction is the decomposition in elementary operation of a business operation. For example, for a cash withdrawal, there is at least a control of the account, a control of the amount of the withdrawals already carried out during the week, a handing-over of cash, and an update of the account.
- ▶ A business operation is equivalent to a hit page into RPT.
- ▶ A business operation is equivalent to more than two CICS® transactions.

Figure 14-3 shows the back-end and hits/second workload.

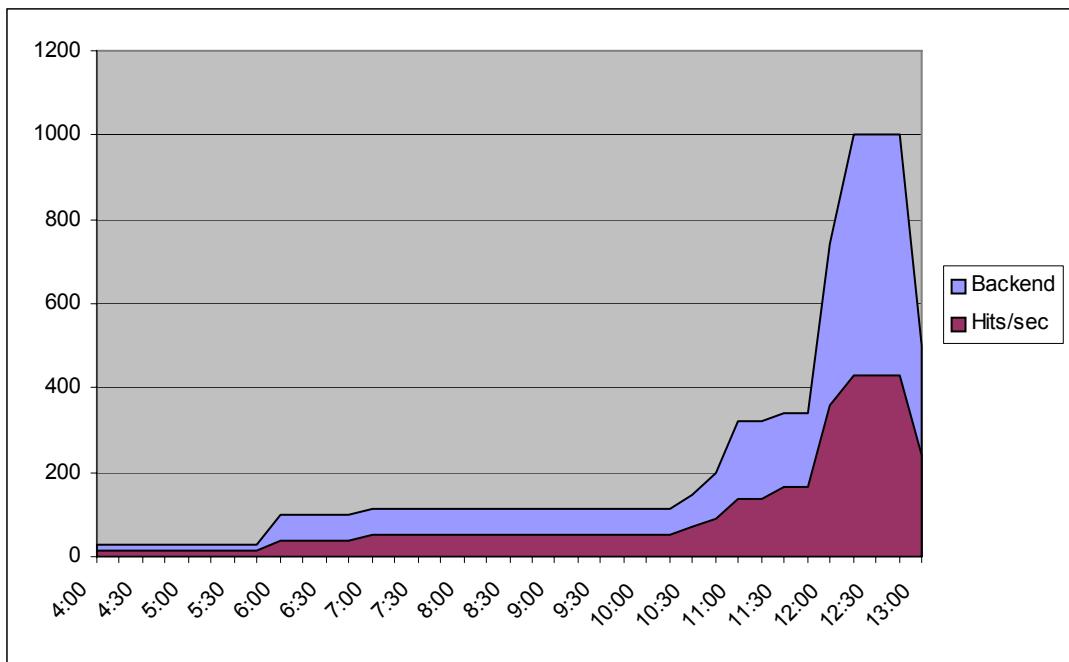


Figure 14-3 Reference of workload to simulate

Table 14-3 shows the hit pages per second for each channel for the same time periods.

Notes:

- ▶ We have two other channels, Gold and Silver, which are ATM channels reserved for VIP.
- ▶ The Process channel is a Teller SOA process.

Table 14-3 Details of hit pages per second for each channel

| Time | Channels | | | | | |
|-------|----------|------|---------|--------|--------|------------------|
| | ATM | Gold | Process | Silver | Teller | Internet Banking |
| 04:00 | 4.5 | 0.5 | N/A | 0.7 | N/A | 6.3 |
| 06:00 | 15 | 1 | N/A | 1.5 | N/A | 22.5 |
| 07:00 | 15 | 1 | N/A | 1.5 | 10 | 22.5 |
| 10:30 | 15 | 1 | 20 | 1.5 | 10 | 22.5 |

| Time | Channels | | | | | |
|--|----------|------|---------|--------|--------|------------------|
| | ATM | Gold | Process | Silver | Teller | Internet Banking |
| 10:30 | 20 | 1.5 | 25 | 2.5 | 14 | 27 |
| 11:00 | 2 | 35 | 4 | 24 | 40 | 2 |
| 11:30 | 3 | 40 | 6 | 30 | 51 | 3 |
| 12:00 | 6 | 58 | 10 | 65 | 126 | 6 |
| In our demonstration, we perform an On/Off Capacity on Demand to our System z9-109 environment, which provisions more processing capacity to our environment. That is, it automatically makes available more processors such as central processors (CPs), zAAPs, and Integrated Facility for Linux (IFLs) for our workload to use. | | | | | | |
| 12:00 | 8 | 65 | 12 | 80 | 155 | 8 |
| 13:00 | 54 | 4 | 60 | 7 | 45 | 70 |

Figure 14-4 and Figure 14-5 show the simulated workload with these values.

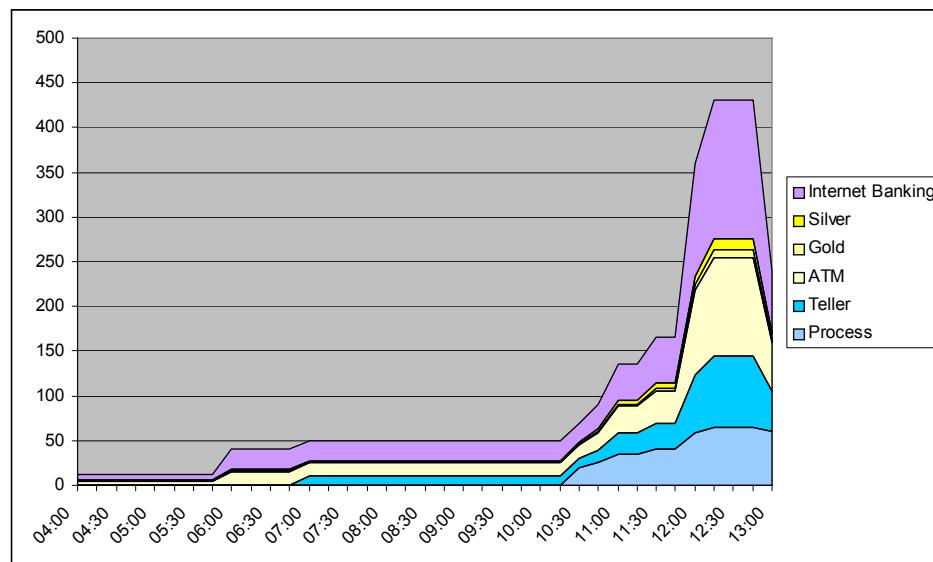


Figure 14-4 Simulated workload

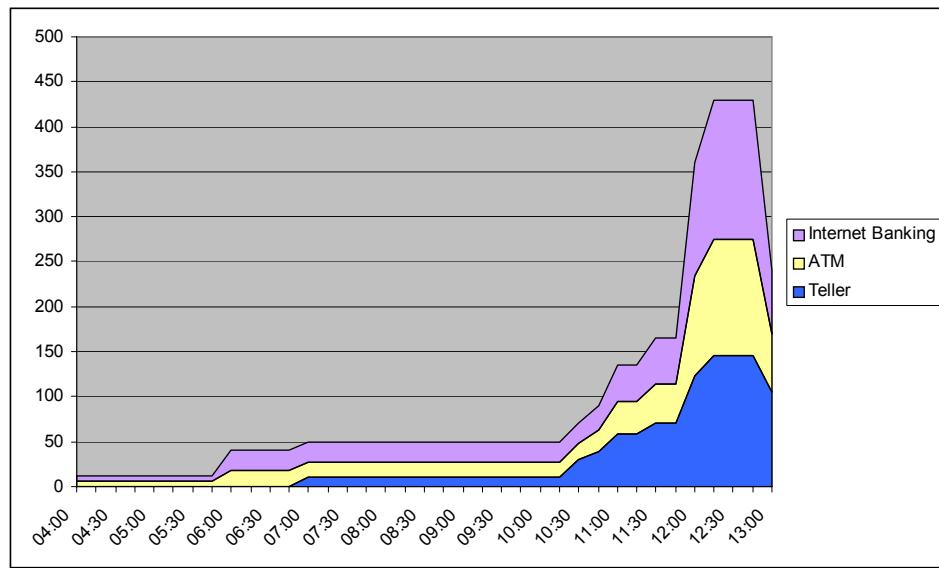


Figure 14-5 Cumulative channels workload

14.3.1 Particular points to consider

Which are the points to consider to set up the showcase:

- ▶ The startup of a channel on demand.
- ▶ The variation of the workload of each channel overtime.
- ▶ RPT possibilities:
 - RPT can:
 - Put a delay before and/or after an action.
 - Put a latency between each action.
 - Create several performance schedules.
 - RPT cannot:
 - Start or stop on demand a test script execution.
 - Change on demand the latency between each action.
 - Run more than one performance schedule at time.
 - Easily change the target IP address and port for injection.

During the implementation of the showcase, these are the main difficulties we found:

- ▶ Creating a unique random number for each virtual user; this is to prevent the possibility that several virtual users might use the same account number (for the implementation, refer to Example 14-4 on page 493).
- ▶ Executing a script, or not (for the implementation, refer to Example 14-6 on page 494).
- ▶ Modifying on demand the cycle of execution (for the implementation, refer to Example 14-7 on page 495).

14.4 Smart Bank showcase implementation

To vary a workload, it is possible to:

- ▶ Increase the number of virtual users
- ▶ Change the latency between each action

For the Smart Bank showcase, we chose the second solution to change the latency between each action.

We decided to implement 3400 virtual users simultaneously.

In the performance schedule, we create a group for each type of action (one for ATM_BI, one for ATM_CW, and so forth).

We require a dynamic workload, as we move from proof-point to proof-point during our day in the life of a bank.

An external tool for RPT, called *RPT Tuning Tool*, was developed, and—associated with custom code—provides this dynamic capability. Otherwise, we would need a different controller for each scenario, because it is not possible to vary think time with the controller (only users can be varied).

The RPT Tuning Tool is based on the management of properties files.

The RPT Tuning Tool generates the suitable properties files and ensures the distribution of them to all Remote Agent Controllers (RAC). Refer to the diagram in Figure 14-6 for details.

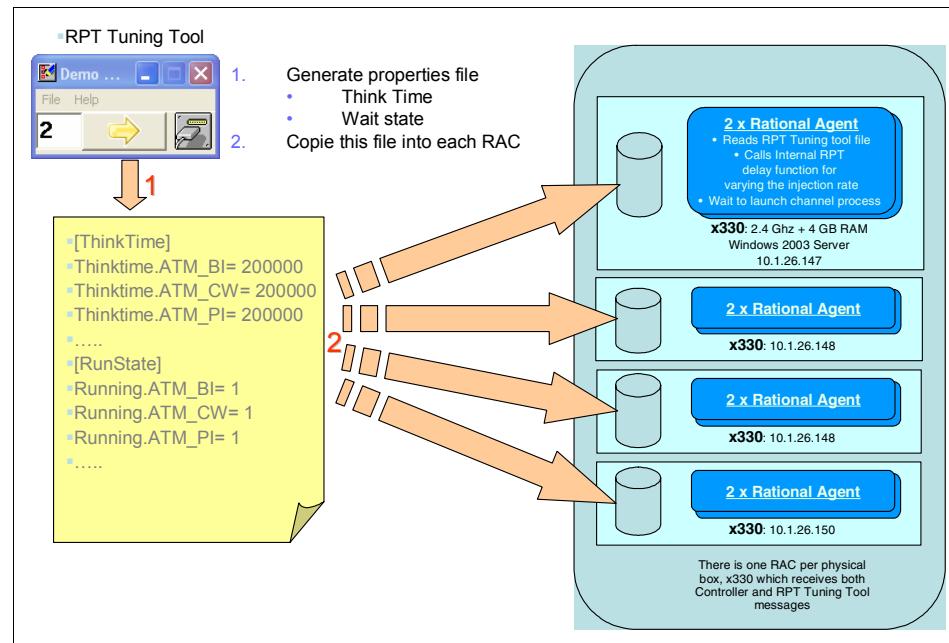


Figure 14-6 RPT Tuning Tool functionality

In RPT we add the custom code as follows:

- ▶ Read the properties file (refer to Example 14-1 on page 487 and Example 14-2 on page 491).
- ▶ Generate a delay (think time) according to the received parameter (refer to Example 14-7 on page 495).
- ▶ Activate (or not) the execution of the test (refer to Example 14-9 on page 497).

To allow the online workload simulation, we have to create a performance schedule that contains as many groups of virtual users as that test, and we allocate for each group a percentage of virtual users (Figure 14-7).

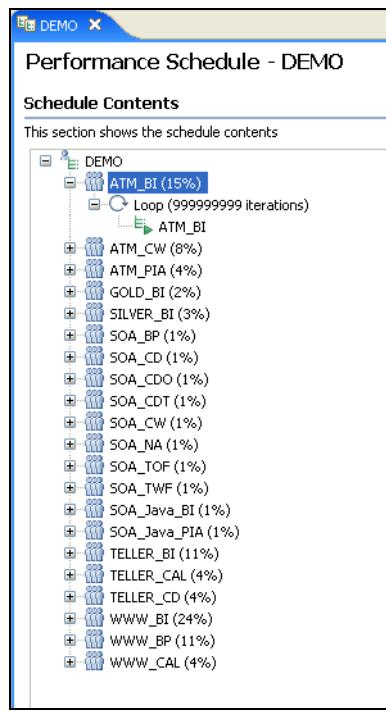


Figure 14-7 Online workload simulation

To calibrate the number of hit/second to be workloaded, we use a manual mode of the RPT Tuning Tool (Figure 14-8). With this dialog, it is possible to easily and quickly modify each test's think time, and to send them on the RAC to have a workload adapted to one of the steps (refer to 14.3, “Workload analysis” on page 479).

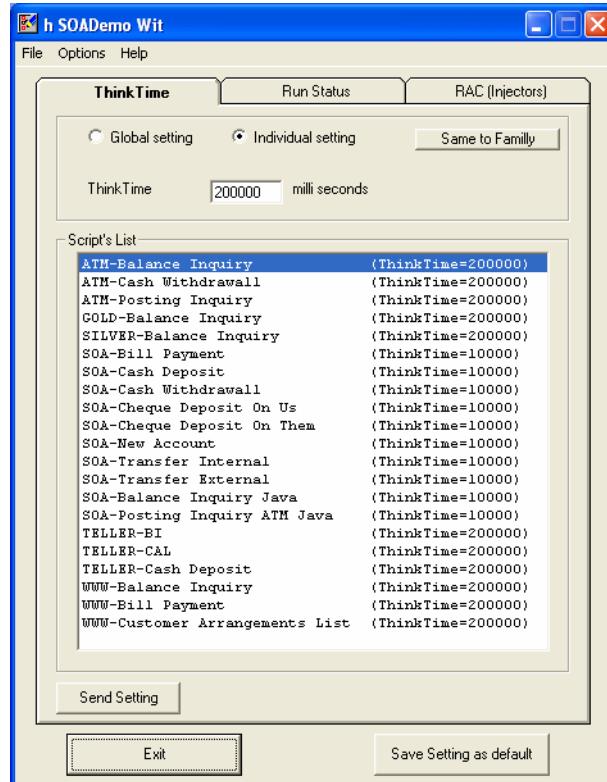


Figure 14-8 RPT Tuning Tool: Think Time data entry

14.4.1 RunProperties custom code

Example 14-1 is used to read the properties file. This code, created as a Java class, is called by Example 14-2 on page 491.

Example 14-1 RunProperties custom code

```
package test.custom;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
```

```

import java.util.Properties;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;

/**
 * @author NICOLASR
 */
public class RunProperties {
    static long ficPropLastModified = 0;
    static Properties prop = new Properties();
    String ficProp = "C:/RPTParams/runProperties.properties";

    /**
     * Instances of this will be created using a String constructor.
     */
    public RunProperties (String fic){
        if (fic != null || fic.equals("")) {
            ficProp = fic;
        }
        if (checkModif()) {
            readProperties();
        }
    }
    public RunProperties (String fic, ITestLogManager tlm){
        tlm.reportMessage("RunProperties .String fic....");
        if (fic != null || fic.equals("")) {
            ficProp = fic;
        }
        if (checkModif(tlm)) {
            readProperties(tlm);
        }
    }
    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public RunProperties (){
        if (checkModif()) {
            readProperties();
        }
    }

    public String get(String key){
        String sret = prop.getProperty(key);
        if (sret == null )
        {
            sret = "";
        }
        return sret.toString();
    }
    public String get(String key, ITestLogManager tlm){

```

```

        String sret = prop.getProperty(key);
        tlm.reportMessage("get("+key+")=" + sret);
        if (sret ==null )
        {
            sret = "";
        }
        return sret.toString();
    }

    public boolean checkModif() {
        boolean modified = false;
        File file = new File(ficProp);
        modified = file.lastModified() > ficPropLastModified;
        return modified;
    }

    public boolean checkModif(ITestLogManager tlm) {
        boolean modified = false;
        File file = new File(ficProp);
        tlm.reportMessage("ficPropLastModified = " +
                           ficPropLastModified);
        modified = file.lastModified() > ficPropLastModified;
        tlm.reportMessage("file.lastModified() = " +
                           file.lastModified());
        tlm.reportMessage("checkModif() = " + modified);
        return modified;
    }

    public boolean readProperties() {
        boolean readed = true;
        FileInputStream in = null;
        try{
            in = new FileInputStream(ficProp);
            if (in != null)
            {
                prop.load(in);
                in.close();
            }
            File file = new File(ficProp);
            ficPropLastModified = file.lastModified();

        }catch (FileNotFoundException e) {
            readed = false;
            e.printStackTrace();
        } catch (IOException e) {
            readed = false;
            e.printStackTrace();
        }
        return readed;
    }
}

```

```

public boolean readProperties(ITestLogManager tlm) {
    boolean readed = true;
    FileInputStream in = null;
    try{
        tlm.reportMessage("Read file " + ficProp);
        in = new FileInputStream(ficProp);
        if (in != null)
        {
            prop.load(in);
            in.close();
        }
        File file = new File(ficProp);
        //tlm.reportMessage("ficPropLastModified = " +
        //                  ficPropLastModified);
        //tlm.reportMessage("file.lastModified() = " +
        //                  file.lastModified());
        ficPropLastModified = file.lastModified();
        tlm.reportMessage("ficPropLastModified = file.lastModified()=" +
                          + ficPropLastModified);

    }catch (FileNotFoundException e) {
        readed = false;
        tlm.reportMessage("RunProperties.readProperties()-"+e);
        tlm.reportMessage("RunProperties.readProperties()-File=" +
                          +ficProp+"=-");
        e.printStackTrace();
    } catch (IOException e) {
        readed = false;
        tlm.reportMessage("RunProperties.readProperties()-"+e);
        tlm.reportMessage("RunProperties.readProperties()-File=" +
                          +ficProp+"=-");
        e.printStackTrace();
    }
    return readed;
}

```

14.4.2 ReadProperties custom code

Example 14-2 is used to read the parameters of the test:

- ▶ Think time, keyword Thinktime.xxxx.
- ▶ Wait status, keyword Running.xxxx.
- ▶ Variables, keywords Var#.xxxx.

For that, the code uses the ScriptName initialized previously. This code uses another custom code shown in 14.4.1, “RunProperties custom code” on page 487.

Example 14-2 ReadProperties custom code

```
package test.custom;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;

/**
 * @author NICOLASR
 */
public class ReadProperties implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ReadProperties() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        ITestLogManager testLogManager = tes.getTestLogManager();
        IDataArea vda = tes.findDataArea( IDataArea.VIRTUALUSER );
        IDataArea tda = tes.findDataArea( IDataArea.TEST );
        String sdebug = "";
        String scriptname = "";
        if (tda.containsKey("ScriptName"))
        {
            scriptname = (String) tda.get("ScriptName");
        }
        //Get properties
        // With Multiple properties files
        RunProperties rp = new RunProperties("C:/RPTParams/" + scriptname
                + "_runProperties.properties");
        //Get debug mode
        sdebug = rp.get("DebugMode");
        vda.put("Debug", sdebug);
        if (sdebug.equals("1"))
        {
            //Get All Parameters for specific script
            testLogManager.reportMessage("Debug = 1");
            vda.put("Thinktime." + scriptname, rp.get("Thinktime."
                    + scriptname, testLogManager));
            vda.put("Running." + scriptname, rp.get("Running."
                    + scriptname, testLogManager));
        }
    }
}
```

```

        vda.put("Var1." + scriptname, rp.get("Var1." + scriptname,
                testLogManager));
        vda.put("Var2." + scriptname, rp.get("Var2." + scriptname,
                testLogManager));
        vda.put("Var3." + scriptname, rp.get("Var3." + scriptname,
                testLogManager));
    }
} else
{
    //Get All Parameters for specific script
    vda.put("Thinktime." + scriptname, rp.get("Thinktime." +
        scriptname));
    vda.put("Running." + scriptname, rp.get("Running." +
        scriptname));
    vda.put("Var1." + scriptname, rp.get("Var1." + scriptname));
    vda.put("Var2." + scriptname, rp.get("Var2." + scriptname));
    vda.put("Var3." + scriptname, rp.get("Var3." + scriptname));
}
return null;
}
}

```

14.4.3 TestName custom code

Example 14-3 is used in the test to know which test is running and thus to read the suitable data into the properties file.

Example 14-3 TestName custom code

```

package test.custom;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author NICOLASR
 */
public class ATM_BI implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ATM_BI() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        IDataArea tda = tes.findDataArea( IDataArea.TEST );
        String sret="ATM_BI";

```

```
        tda.put("ScriptName", sret );
        return sret;
    }
}
```

14.4.4 Account custom code

Example 14-5 is used to have a unique account number. To do this, the random number used is not the standard math class, but the virtual user's class. This ensures having a different number for each virtual user (Example 14-4).

Example 14-4 Get virtual user Random

```
IVirtualUserInfo ivuser = (IVirtualUserInfo)
    tes.findDataArea(IDataArea.VIRTUALUSER).get(IVirtualUserInfo.KEY);
Random ivrand = ivuser.getRandom() ;
```

Example 14-5 Account custom code

```
package test.custom;

import java.util.Random;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.IVirtualUserInfo;

/**
 * @author NICOLASR
 */
public class Account implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    private String AccountNumber;
    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public Account() {
    }
    public String exec(ITestExecutionServices tes, String[] args) {
        IVirtualUserInfo ivuser = (IVirtualUserInfo)tes
            .findDataArea(IDataArea.VIRTUALUSER).get(IVirtualUserInfo.KEY);
        Random ivrand = ivuser.getRandom() ;
        long acc = (long) ((1 + ivrand.nextInt(10798406)));
        AccountNumber = String.valueOf(acc + 800000000L);
        return AccountNumber;
    }
}
```

14.4.5 RunState custom code

Example 14-6 is used to allow the conditional execution of the URL call of the test. For that, the code uses the keyword Running.xxxx, in which xxxx is the name of the script that is running.

Example 14-6 RunState custom code

```
package test.custom;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author NICOLASR
 */
public class RunState implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public RunState() {
    }

    /**
     * For description of ICustomCode2 and ITestExecutionServices
     * interfaces, see Javadoc located at
     * <INSTALL_DIR>/rpt_prod/eclipse/plugins/
     * com.ibm.rational.test.lt.kernel_<PROD_VERSION>/pubjavadoc
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        IDataArea vda = tes.findDataArea( IDataArea.VIRTUALUSER );
        IDataArea tda = tes.findDataArea( IDataArea.TEST );
        String scriptname = "";
        if (tda.containsKey("ScriptName"))
        {
            scriptname = (String) tda.get("ScriptName" );
        }
        return (String) vda.get("Running." + scriptname );
    }
}
```

The use of the return code is shown in Figure 14-9 on page 497.

14.4.6 DoThinkTime custom code

Example 14-7 is used to dynamically generate a delay between each loop action. For that, the code uses the keyword ThinkTime.xxxx, in which xxxx is the name of the script that is running.

Important:

- ▶ To pace the rate of test execution, it is necessary to insert a think time delay transaction that contains this custom code (refer to Figure 14-9 on page 497).
- ▶ The interfaces of objects KAction and KDelay are not documented.

Example 14-7 DoThinkTime custom code

```
package test.custom;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.action.impl.KAction;
import com.ibm.rational.test.lt.kernel.action.impl.KDelay;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;

/**
 * @author NICOLASR
 */
public class DoThinkTime implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public DoThinkTime() {
    }

    /**
     * For javadoc of ICustomCode2 and ITestExecutionServices
     * interfaces, select 'Help Contents' in the
     * Help menu and select 'IBM Rational Performance Tester TES'.
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        ITestLogManager testLogManager = tes.getTestLogManager();
        IDataArea vda = tes.findDataArea( IDataArea.VIRTUALUSER );
        IDataArea tda = tes.findDataArea( IDataArea.TEST );
        int multi = 1; // 1 => iThinkTime in milli seconds
        int iThinkTime = 10000; // default 10s
        String scriptname = "";
        String debug = "";
```

```

        if (tda.containsKey("ScriptName"))
        {
            scriptname = (String) tda.get("ScriptName" );
        }
        if (vda.containsKey("Debug"))
        {
            debug = (String) vda.get("Debug");
        }
        if (vda.containsKey("Thinktime." + scriptname ))
        {
            iThinkTime = Integer.parseInt((String) vda.get("Thinktime." +
scriptname ));
        }
        if (debug.equals("1" ))
        {
            testLogManager.reportMessage("Thinktime." + scriptname + " = "
+ iThinkTime + " milli seconds");
        }
        if (tes instanceof KAction) {
            int thinkTime = iThinkTime * multi;
            KAction action = (KAction) tes;
            KDelay delay = new KDelay(action.getParent(),
                action.getName()+"-Delay",
                action.getId()+"-Delay", thinkTime);
            action.getParent().add(delay);
        }
        return null;
    }
}

```

14.4.7 Test personalization

In this section we consider some aspects of test personalization.

Use of the custom code in the tests

For each test it is necessary to add these custom code examples:

- ▶ Refer to 14.4.3, “*TestName* custom code” on page 492.
- ▶ Refer to 14.4.2, “*ReadProperties* custom code” on page 490.
- ▶ Refer to 14.4.5, “*RunState* custom code” on page 494.
- ▶ Refer to 14.4.6, “*DoThinkTime* custom code” on page 495.

Figure 14-9 shows an example of a complete test with its custom code, its conditional run, and its dynamic think time. The custom code 14.4.4, “*Account* custom code” on page 493 was added to make substitutions into the URL.

Performance Test - ATM_BI

Test Contents
This section shows the test contents

Test Element Details

Conditional(IF) Block
Condition:
Type text to compare or select a data source object from a combobox
First operand: Custom code: test.custom.RunState Go To
Selected data type: **Custom Code**
Operator: Equals
Second operand: 1
Selected data type: **text value**

Add Insert Remove Up Down Prev Next Negate the operator (i.e. NOT(op)) Case-sensitive comparison

Figure 14-9 Complete test

14.5 RPT V7 architecture for workload injection

Figure 14-10 shows a view of the architectural solution of the workload. This configuration lets you inject more transactions than a z9 server could support.

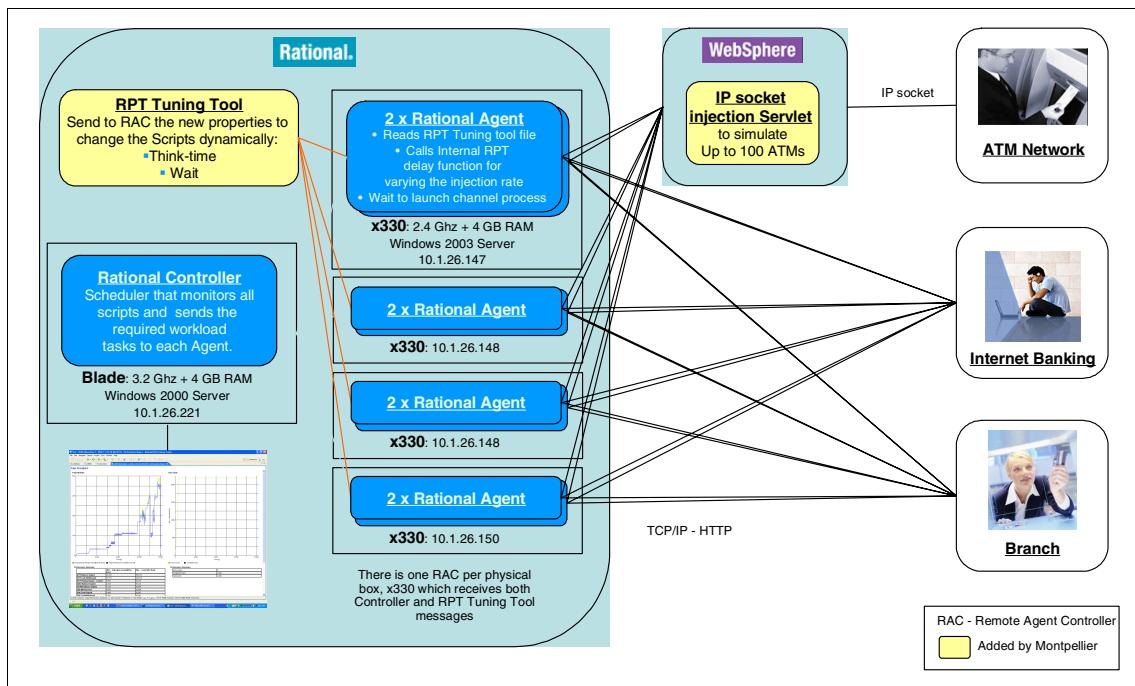


Figure 14-10 Workload injection architecture

We defined two Rational agent location objects using different host names for each IBM xSeries 330 system, because these systems have 4 GB of memory and each test agent's JVM is limited to a maximum of 1.8 GB memory.

To permit the use of two agent locations on the same system, you must follow these steps:

- ▶ Define an alias into the HOST file of the Rational controller.
- ▶ During the creation of the locations:
 - Indicate the alias as an IP address.
 - Indicate a different deployment directory for each location.

14.6 Conclusion

Figure 14-11 shows the result of a dynamic workload generation during a typical Smart Bank showcase. This workload is done with eight agents and can push 1200 back-end transactions/sec. (The back-end transactions are the real transactions on the level of the server z9. A simple test action can generate several actions on the server.)

With this principle of dynamic workload, it is possible to know which is the peak load supported by the application or the server to be tested.

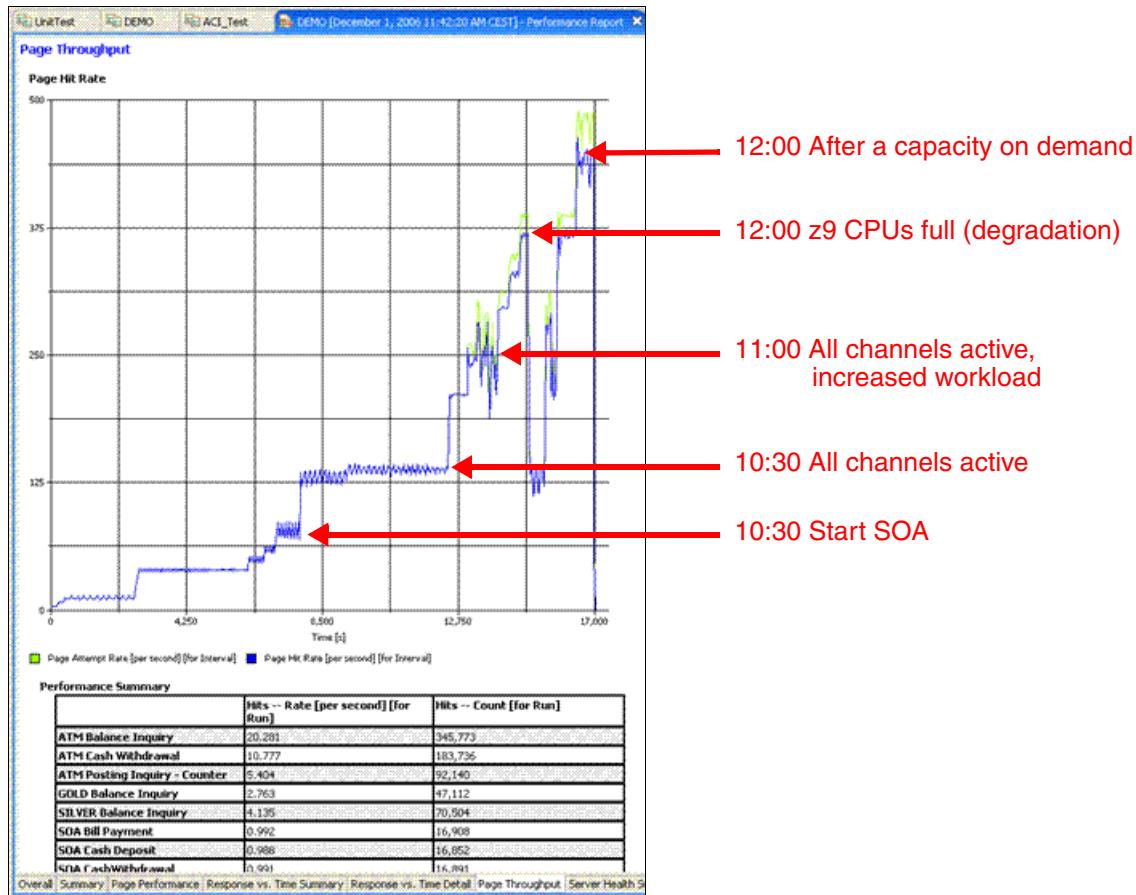


Figure 14-11 Typical Smart Bank showcase



Part 3

Appendices



A

Installation and licensing

In this Appendix we describe the product installation and the licensing requirements.

Product installation

The IBM Rational Performance Tester product has a distributed architecture and therefore has separate install procedures for the product console and the remote test agent. There are two sets of installation media: one for the product console (three disks) and one for the test agent (two disks). When installing from the media, the startup program is called the launchpad. The product install options are listed and initiated from the launchpad (Figure A-1).



Figure A-1 Performance Tester Console Install Launchpad

From the launchpad, you select the option to install Rational Performance Tester and it tries to find an existing version of the Installation Manager. If one is not present on the system, then a copy of the Installation Manager is installed.

For running multi-user performance tests, your Rational Performance Tester product checks out “Virtual Tester” license packs for all but five of the multiple users being simulated. These licenses are considered “floating” licenses that are available to any installation on the network and can be shared (not at the same time) among projects or users of Rational Performance Tester and Rational Robot products.

To support this product licensing model, you have to access an existing license server hosting these licenses or install one for use with this product. There is a Launchpad link available to install the Rational License Server for this purpose. This installation is covered in a subsequent section of this chapter.

Installing the Installation Manager

All of the Rational Eclipse-based, Version 7 desktop products share a common installation mechanism. There is an Installation Manager that provides the ability to install desktop products from CDs or from a local or remote directory or Web server.

Step 1. From the first page of the InstallShield Wizard for the Installation Manager (Figure A-2), click *Next*.

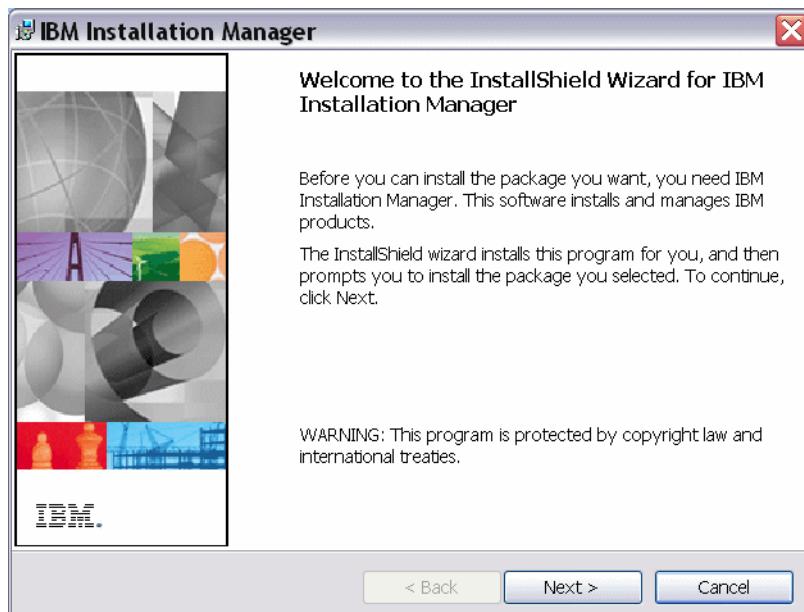


Figure A-2 Installation Manager Installation Wizard - Page 1

Step 2. Accept the license agreement terms (Figure A-3) and click *Next*.

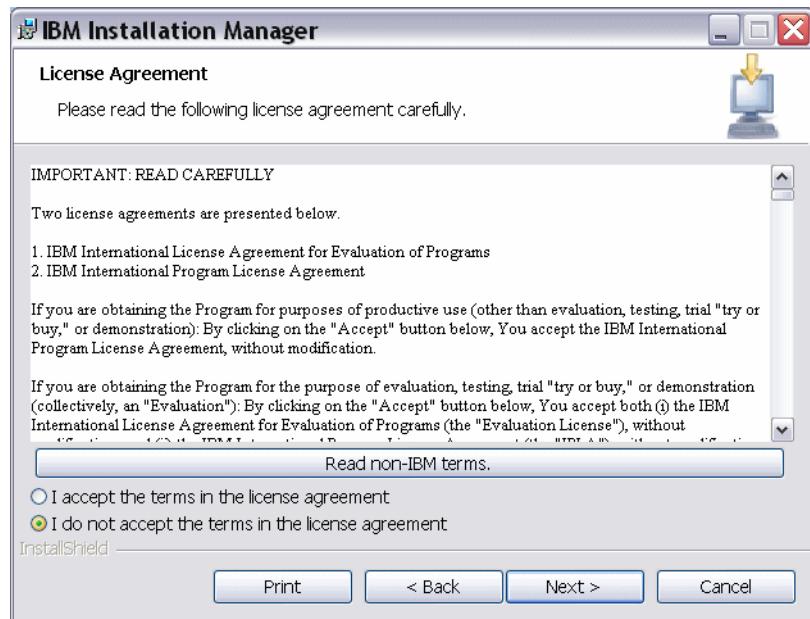


Figure A-3 Installation Manager Installation Wizard - Page 2

Step 3. Change the installation directory if you want (Figure A-4) and click *Next*.

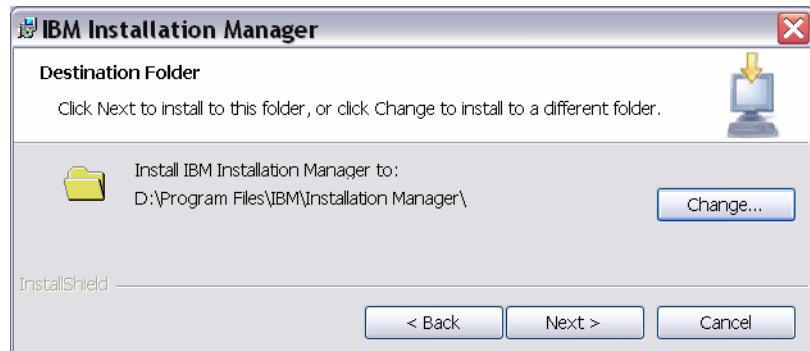


Figure A-4 Installation Manager Installation Wizard - Page 3

Step 4. Now click *Install* to start the actual install (Figure A-5).

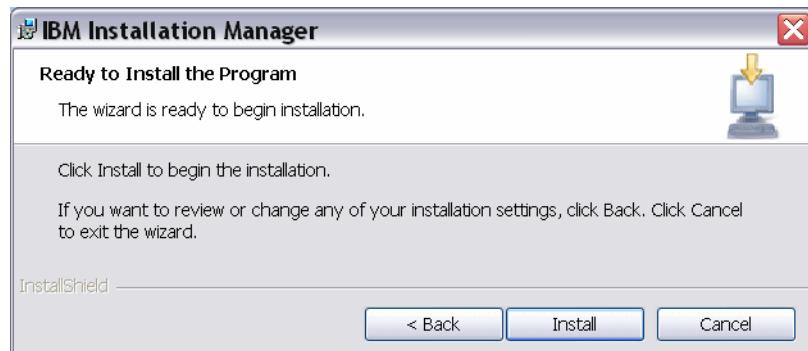


Figure A-5 Installation Manager Installation Wizard - Page 4

Step 5. You can monitor the installation in the status bar (Figure A-6).



Figure A-6 Installation Manager Installation Wizard - Page 5

Step 6. You then get an indication of a successful install (Figure A-7).

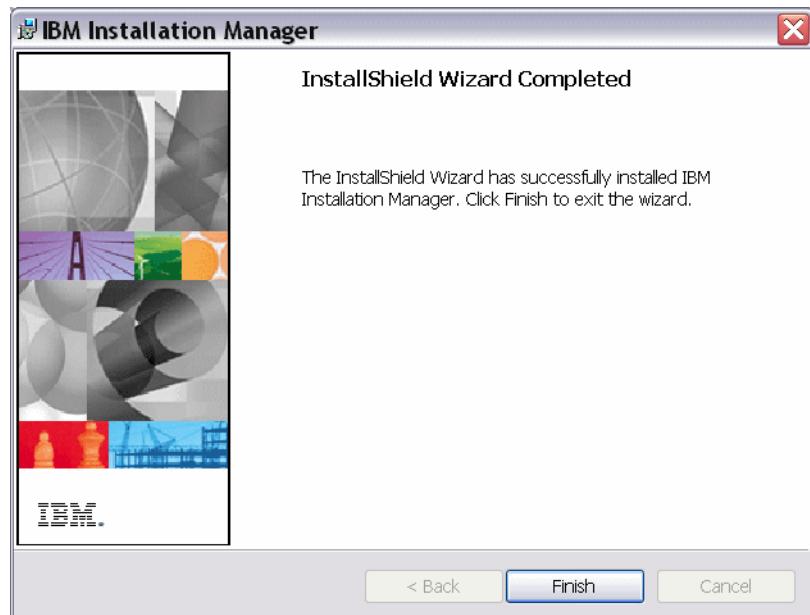


Figure A-7 Installation Manager Installation Wizard - Page 6

If this installation was part of installing a Rational desktop product, then the Installation Manager will come up with an indication of what products are available for installation from the configured repositories.

The Installation Manager can be configured to reference product images from repositories residing on the Internet, a local Web server, a network shared directory, or a local file system. Using the *File → Preferences* menu item, you can set up to whichever repositories you have access. If installing product from the CD images and using the Launchpad application, the repository entries are preset properly to request the CDs as they are needed.

Installing Rational Performance Tester

If you have an installed Installation Manager, you have to start it and click on the Install icon. If you do not, the Installation Manager will be installed from the product installation Launchpad application so that the product can then be installed. After it is installed, the Installation Manager will come up automatically on the Install Packages screen. The available packages will be listed and the product you selected from the Launchpad will be pre-selected from the list.

Step 1. The preselected package installation sequence will begin when you click *Next* (Figure A-8).

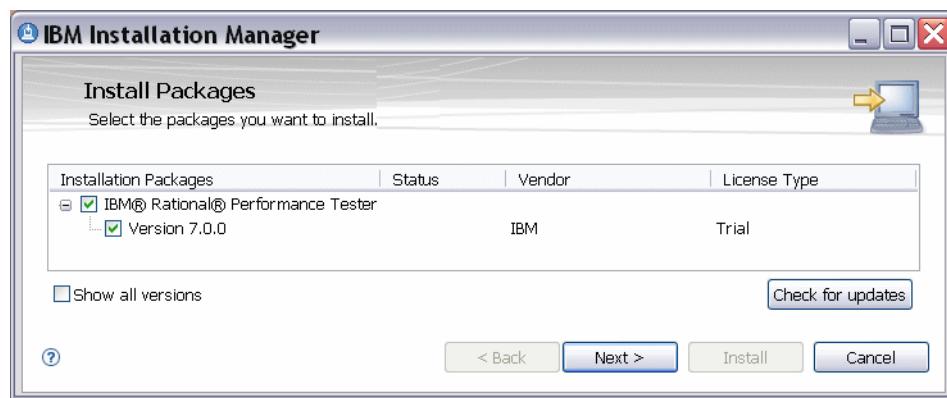


Figure A-8 Performance Tester Installation - Package List

Step 2. Select *I accept the terms in the license agreements* and click *Next* (Figure A-9).

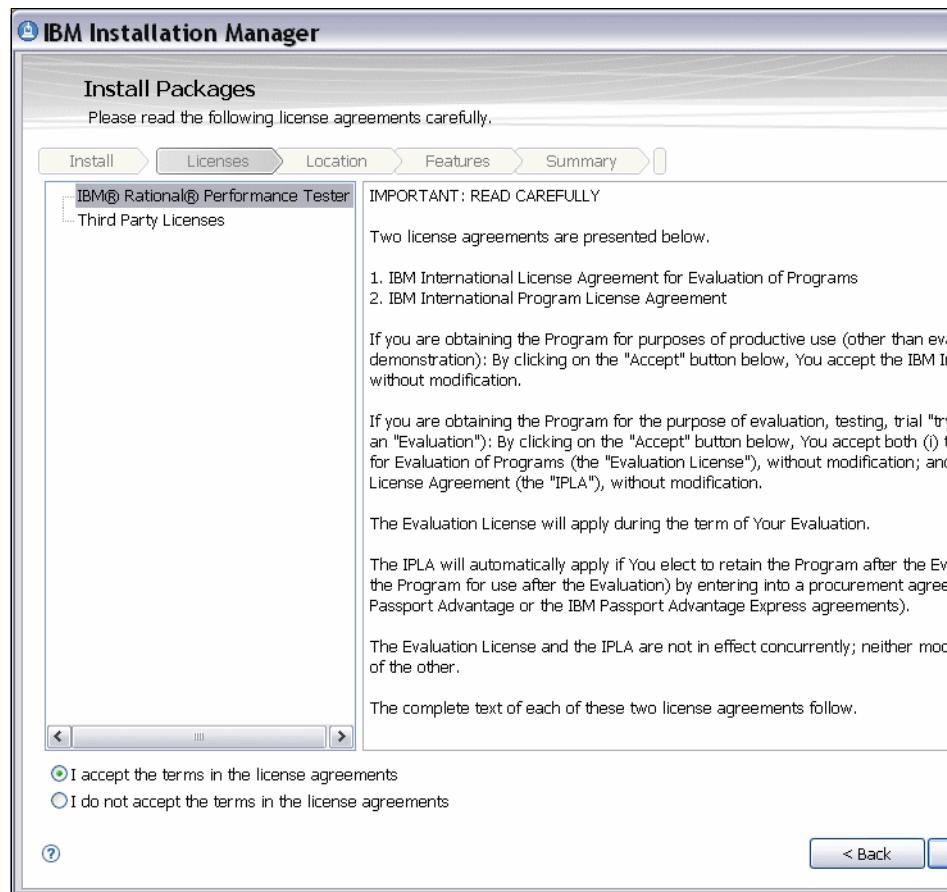


Figure A-9 Performance Tester Installation - License Agreement

Step 3. Browse to the directory where you want the shared desktop product components to be stored and click *Next* (Figure A-10).

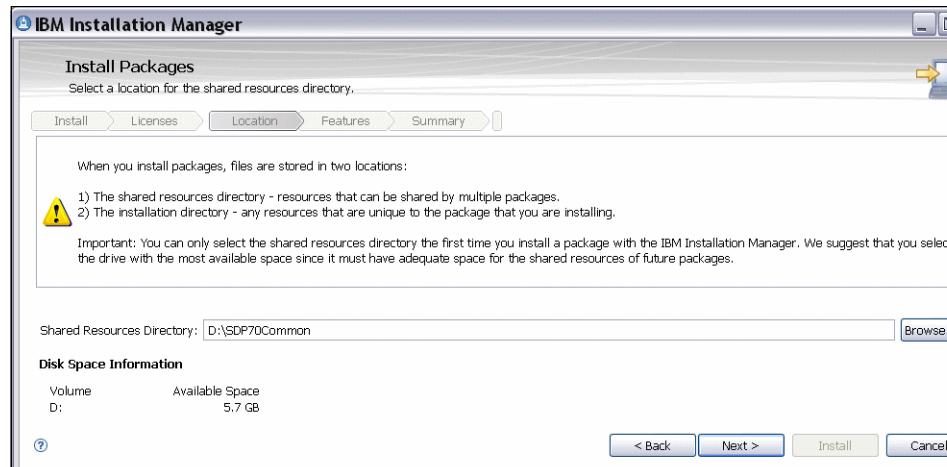


Figure A-10 Performance Tester Installation - Shared Resources

Step 4. Browse to an empty directory to install the package and click *Next* (Figure A-11).

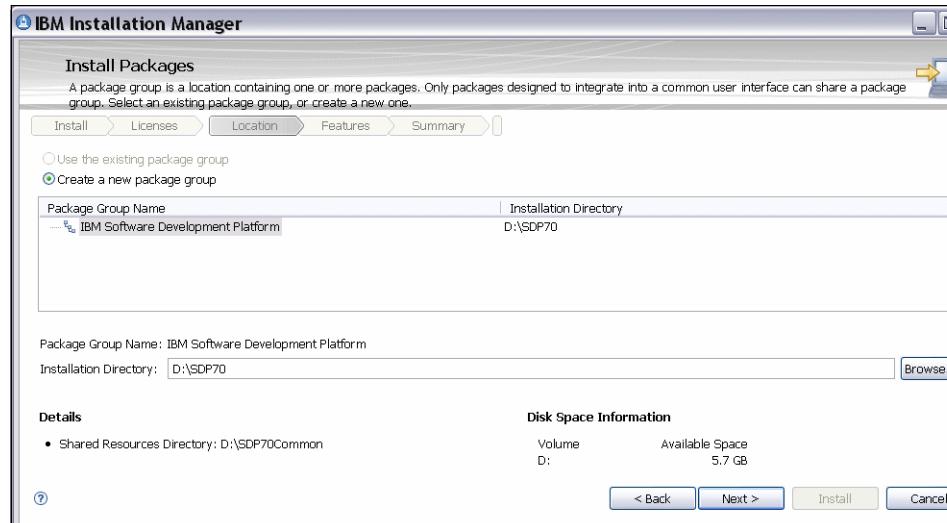


Figure A-11 Performance Tester Installation - Package Directory

Step 5. If you want to lay down a new version of Eclipse for this package, just click *Next*. Otherwise, select *Extend an existing Eclipse*, browse to the installation directory of the Eclipse and JVM that you want to extend, and then click *Next* (Figure A-12).

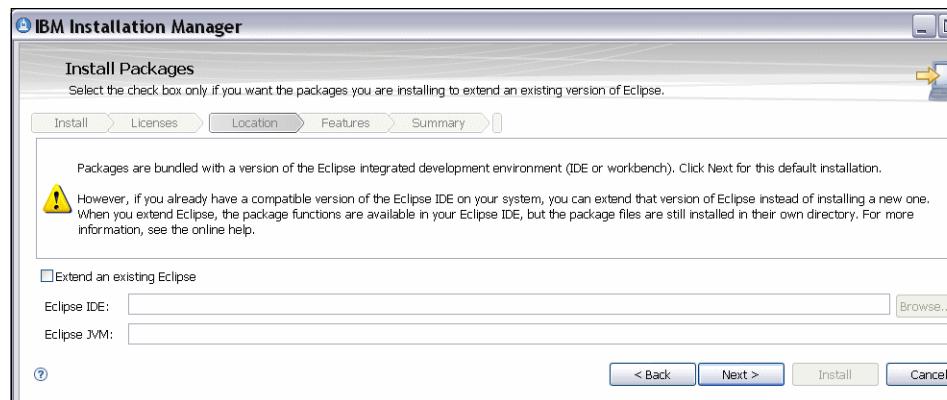


Figure A-12 Performance Tester Installation - Share Eclipse

Step 6. Select any additional languages that you want installed and click *Next* (Figure A-13).

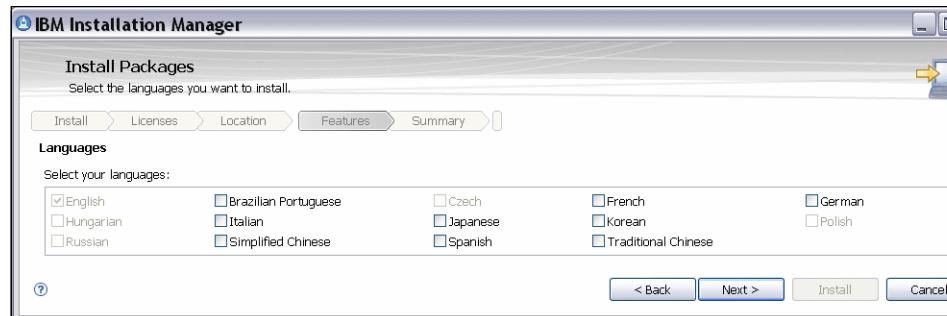


Figure A-13 Performance Tester Installation - Language Packs

Step 7. Select any optional features you wish to install and click *Next* (Figure A-14).

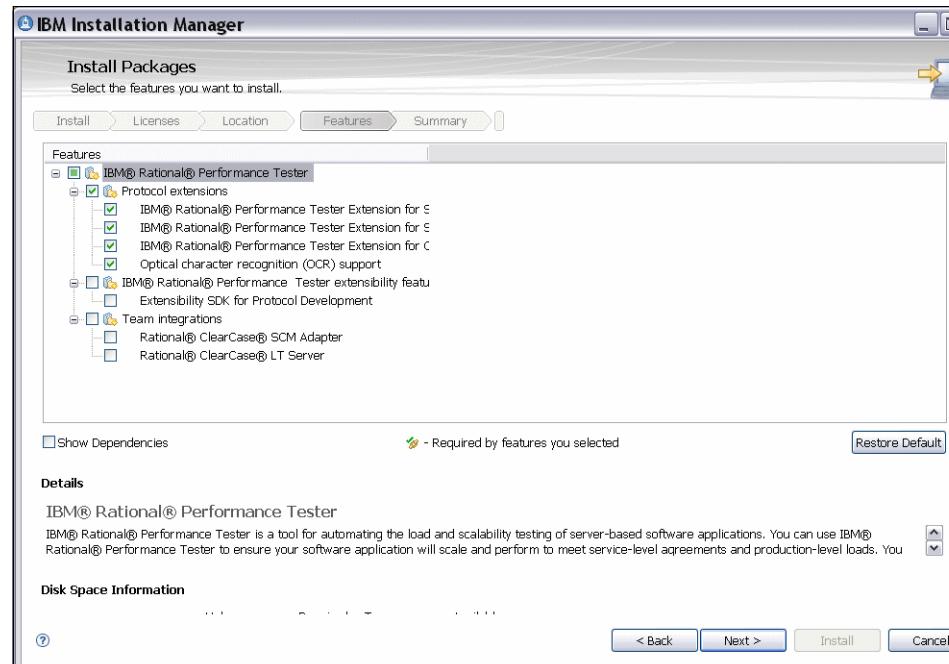


Figure A-14 Performance Tester Installation - Install Features

Step 8. Select the level of security you require for the Test Controller. By default, any computer can connect and use your computer as a Test agent. If this is satisfactory in your environment, it is the best choice. If you want to restrict test runs on your computer to only those tests initiated from your machine, you can customize the settings and select *This computer only*. The third choice is to specify a list of allowed computers (Figure A-15). Click *Next*.

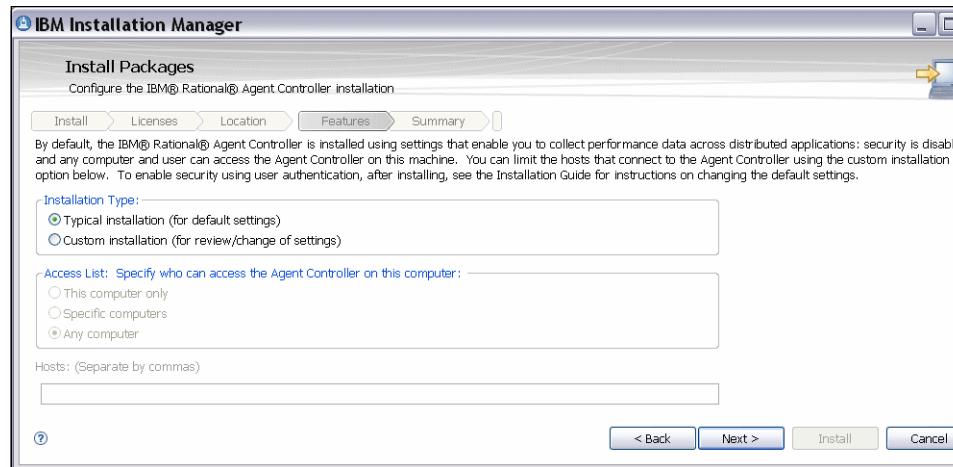


Figure A-15 Performance Tester Installation - Test Controller

Step 9. Review the features to be installed for completeness and click *Install* (Figure A-16).

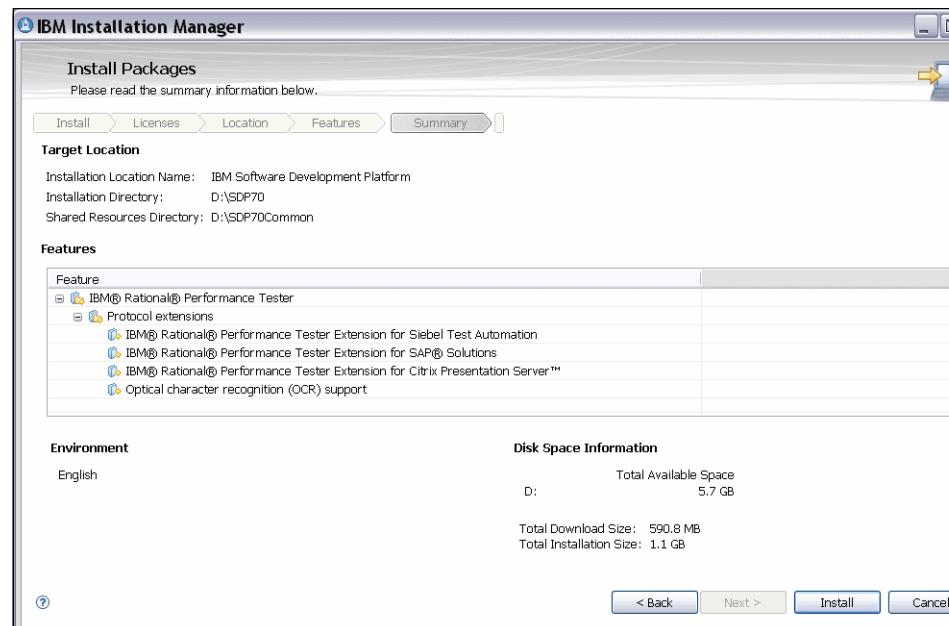


Figure A-16 Performance Tester Installation - Feature Summary

Step 10. Wait for the Installation Manager to complete the install operation (Figure A-17).

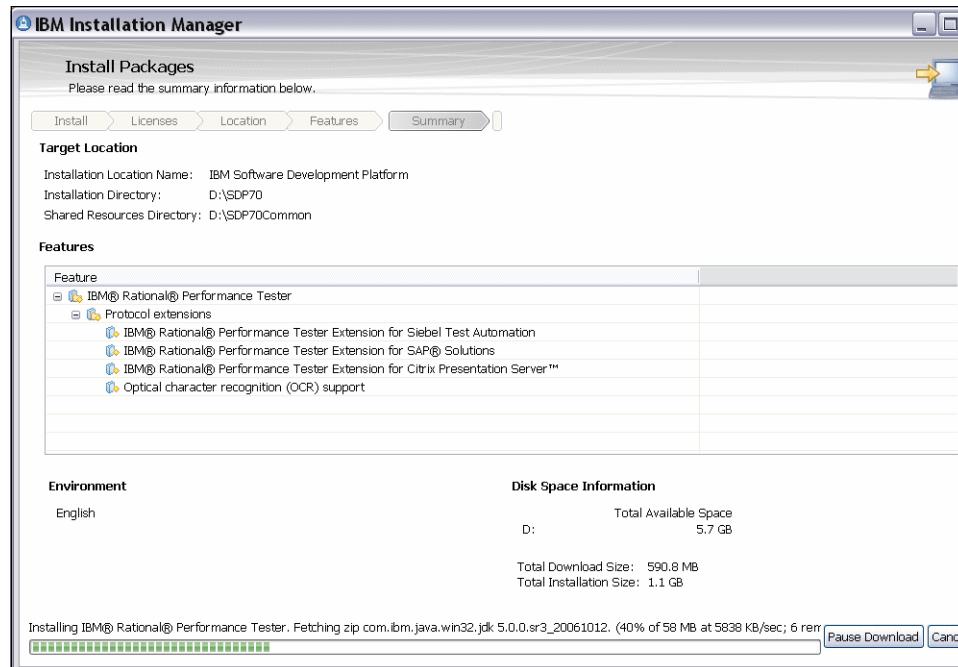


Figure A-17 Performance Tester Installation- Install Status

Step 11. Insert disk 2 when requested, or browse to the installation directory (Figure A-18).

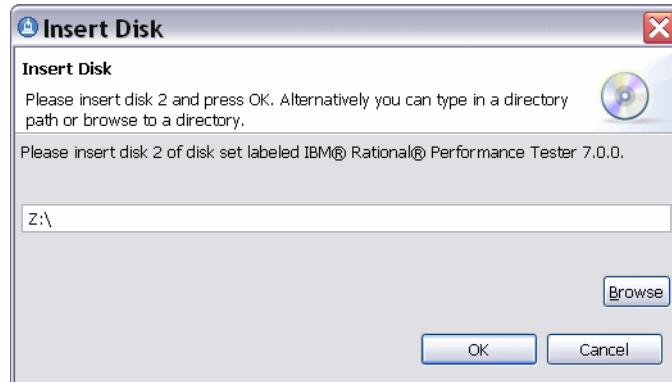


Figure A-18 Performance Tester Installation - Insert Disk dialog for disk 2

Step 12. Insert disk 3 when requested, or browse to the installation directory (Figure A-19).

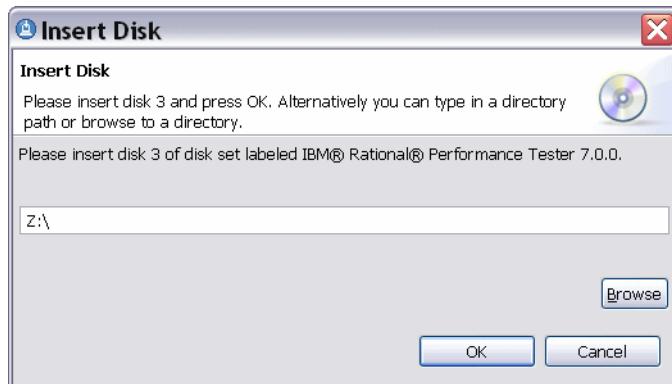


Figure A-19 Performance Tester Installation - Insert Disk dialog for disk 3

Step 13. When the install is complete, you will see a final status message indicating the success or failure of the product installation. Click *Finish* (Figure A-20).

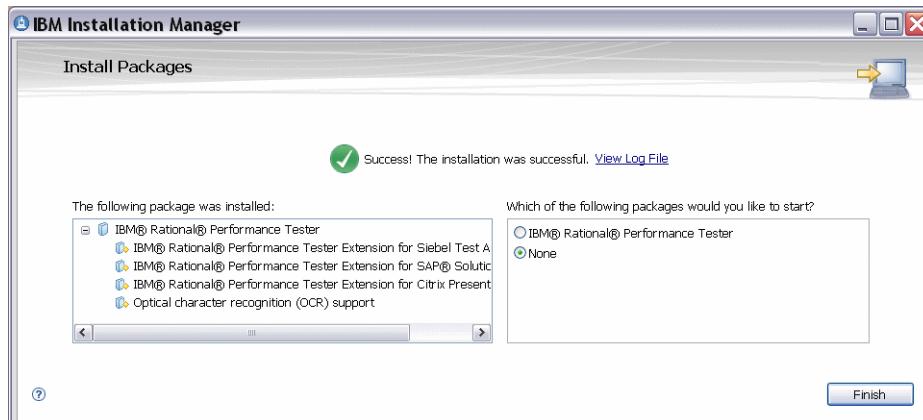


Figure A-20 Performance Tester Installation- Install Package Status

If for any reason the Installation Manager is unable to install the requested package due to conflicting or non-supported components, this page will have a big red X and indicate in the installation log why it failed.

Installing Rational License Server

Step 1. When you click the link to install the Rational License Server, you receive a prompt to insert the special CD for the License Server. When you have inserted it in your CD drive, click *OK* (Figure A-21).

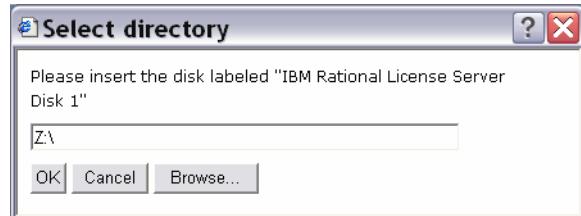


Figure A-21 License Server - Insert Disk dialog

Step 2. From the License Server LaunchPad, select *Install IBM Rational License Server* (Figure A-22).

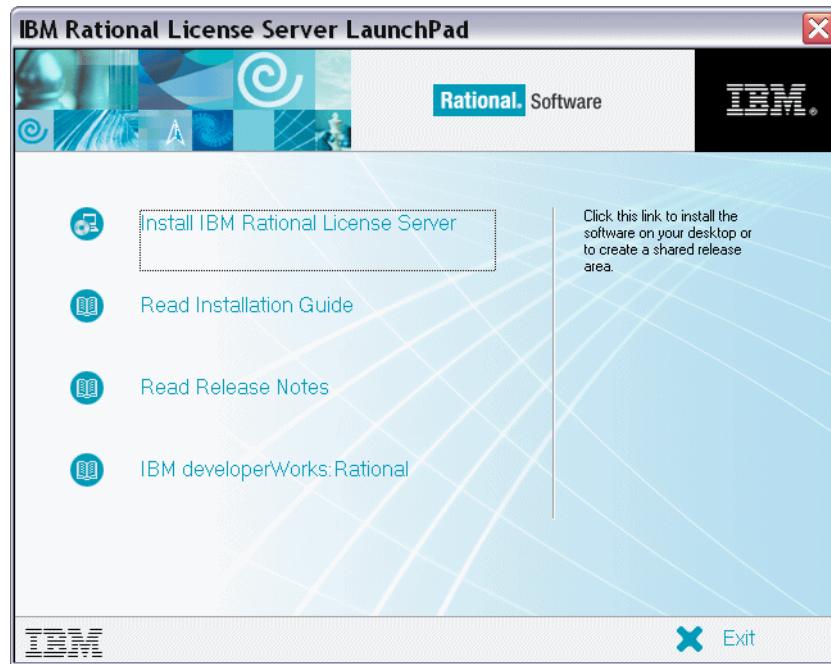


Figure A-22 License Server - LaunchPad

Step 3. Click *Next* at the general welcome screen (Figure A-23).

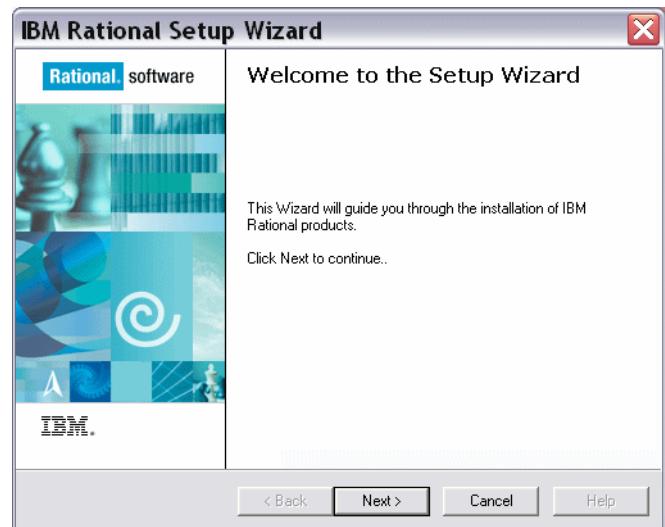


Figure A-23 License Server Setup Wizard - Welcome

Step 4. Click *Next* at the License Server welcome screen (Figure A-24).



Figure A-24 License Server Setup Wizard - Welcome

Step 5. Close all other applications, turn off your anti-virus software, and click *Next* (Figure A-25).

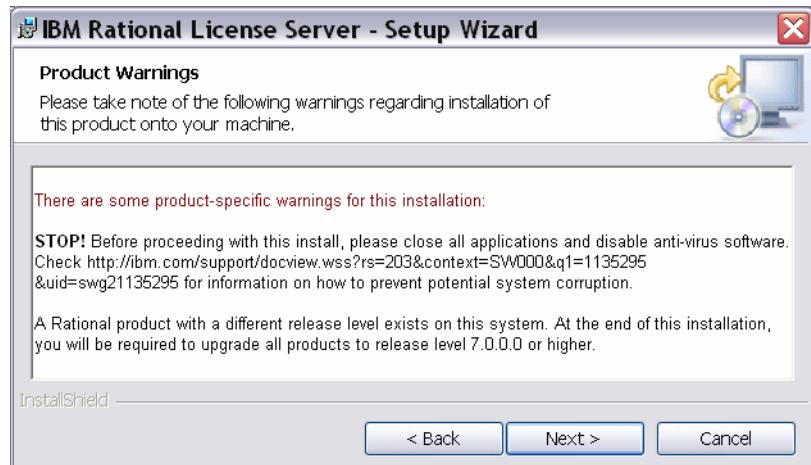


Figure A-25 License Server Setup Wizard - Warning

Step 6. Read and agree with the terms of the license agreement and click *Accept* (Figure A-26).



Figure A-26 License Server Setup Wizard - License Agreement

Step 7. Set the installation location for your license server and click *Next* (Figure A-27). If you have already installed the product, this will be unchangeable.



Figure A-27 License Server Setup Wizard - Destination folder

Step 8. Leave the default setting and click *Next* (Figure A-28).



Figure A-28 License Server Setup Wizard - Feature Selection

Step 9. Click *Install* to begin the installation (Figure A-29).



Figure A-29 License Server Setup Wizard - Install

Step 10. Wait for the installation to complete (Figure A-30).

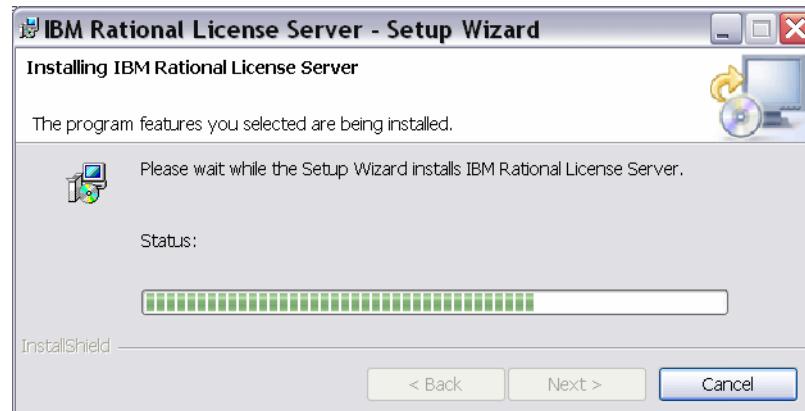


Figure A-30 License Server Setup Wizard - Install Progress

Step 11. The license server is now installed; click *Finish* to exit (Figure A-31).



Figure A-31 License Server Setup Wizard - Install Completion

Configuring the license server and entering license keys are discussed in the product licensing section that follows.

Installing Rational Performance Tester Agent

The Rational Performance Tester Agent should be installed on those lab systems that are to be used as performance test agents (to drive the virtual user sessions). The Test Agent serves as the remote agent for Performance Tester during test playback. The Test Agent should also be installed on those Web application server systems to be monitored for transaction details.

For the transaction breakdown feature to work, the data collection process must be running on all Test Agents whether they are resident on systems used for driving load or on server systems. In addition, the data collection process must be running on the Performance Tester console system.

When installing from the media, the startup program is called the launchpad. The product install options are listed and initiated from the launchpad (Figure A-32).

Step 1. Click on the link *Install IBM Rational Performance Tester Agent*.



Figure A-32 Performance Tester Agent - Install Launchpad

Step 2. Select the package and version to install and click *Next* (Figure A-33).

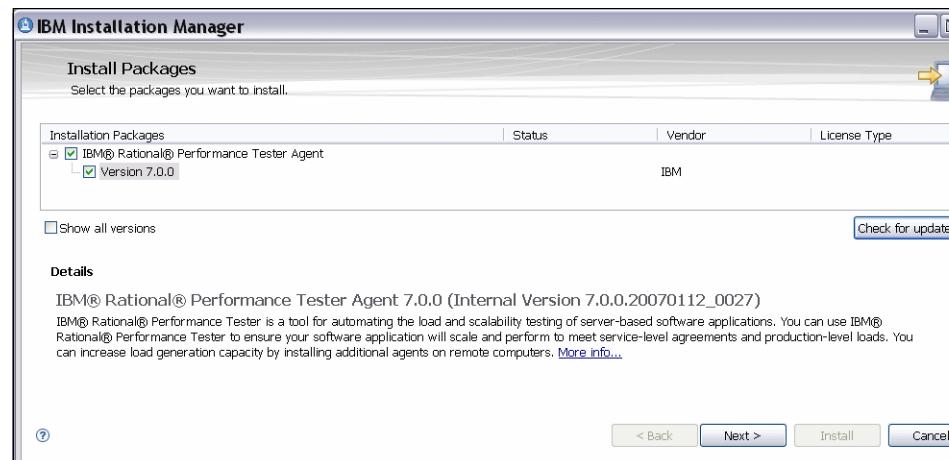


Figure A-33 Performance Tester Agent - Package List

Step 3. Select *I accept the terms of the license agreement* and click *Next* (Figure A-34).

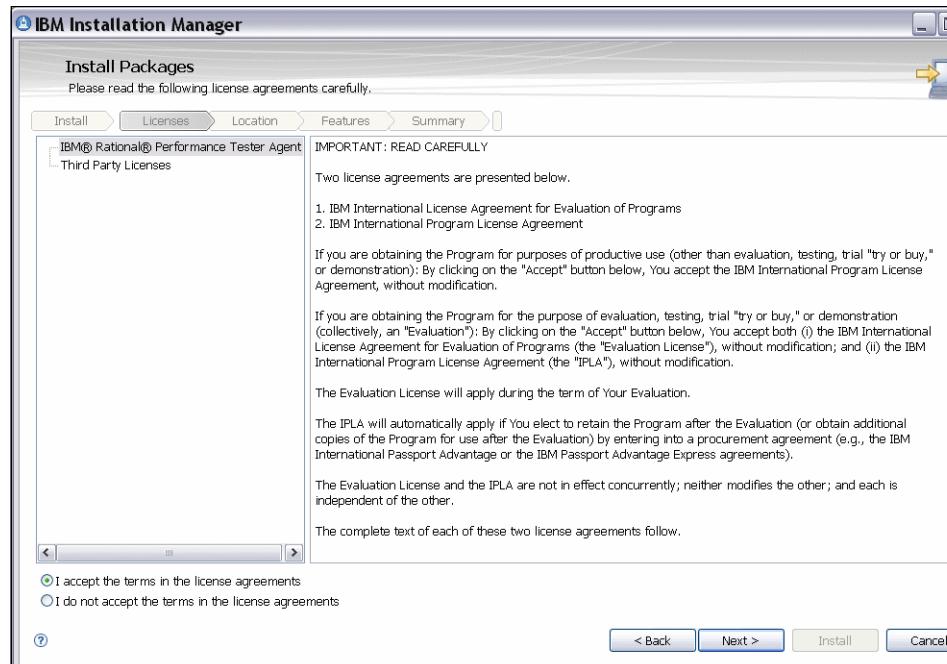


Figure A-34 Performance Tester Agent - License Agreement

Step 4. Browse to the directory where you want the shared desktop product components to be stored and click *Next* (Figure A-35).

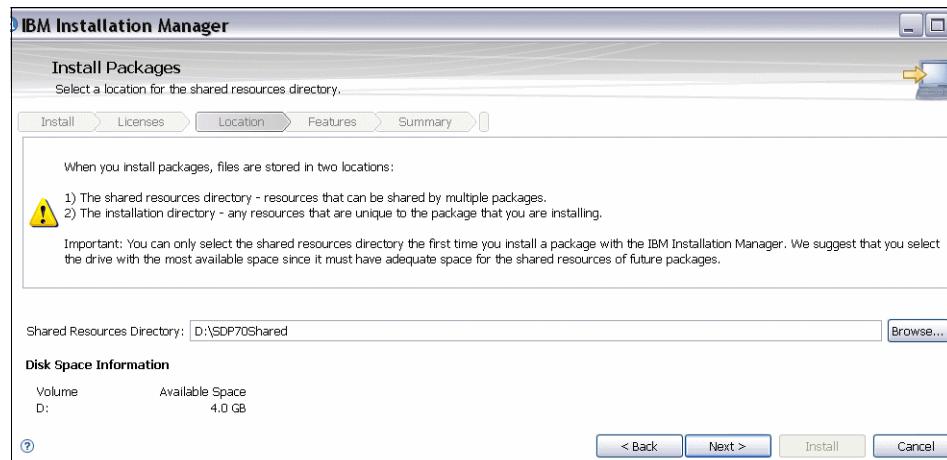


Figure A-35 Performance Tester Agent - Shared Resources

Step 5. Browse to an empty directory to install the package and click *Next* (Figure A-36).

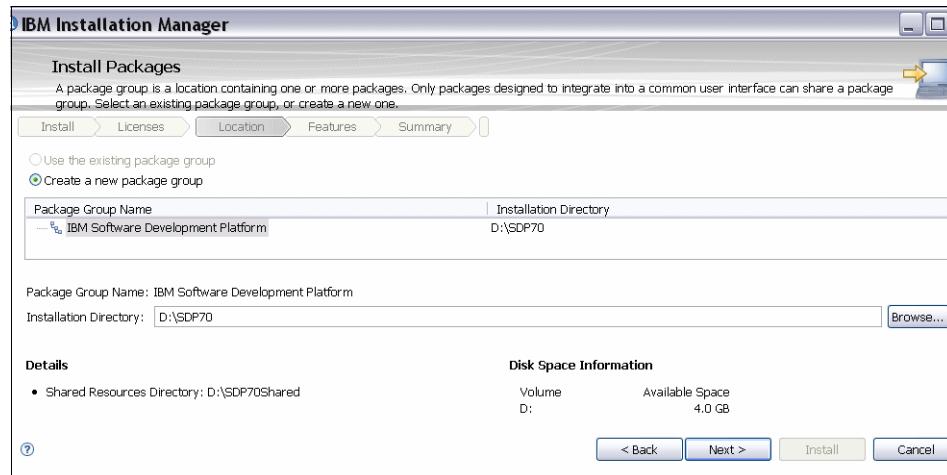


Figure A-36 Performance Tester Agent - Package Directory

Step 6. Click *Next* because the Test Agent does not share an Eclipse instance (Figure A-37).

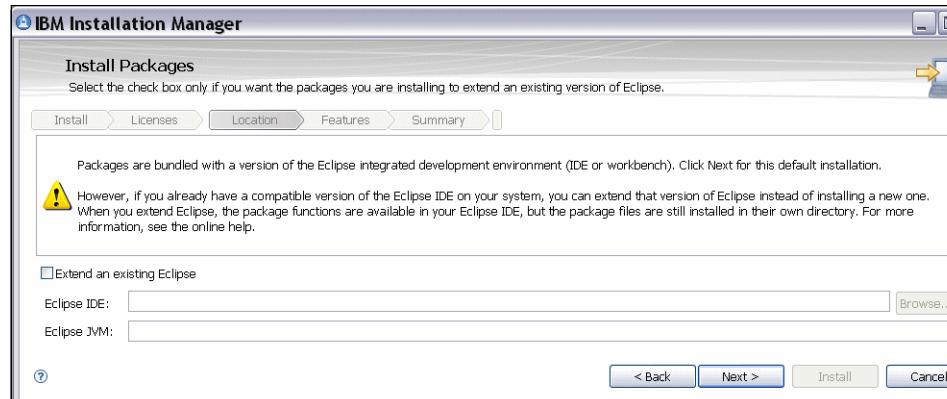


Figure A-37 Performance Tester Agent - Share Eclipse

Step 7. Select any additional languages that you want installed and click *Next* (Figure A-38).



Figure A-38 Performance Tester Agent - Language Packs

Step 8. Click *Next* to install the preselected features (Figure A-39).

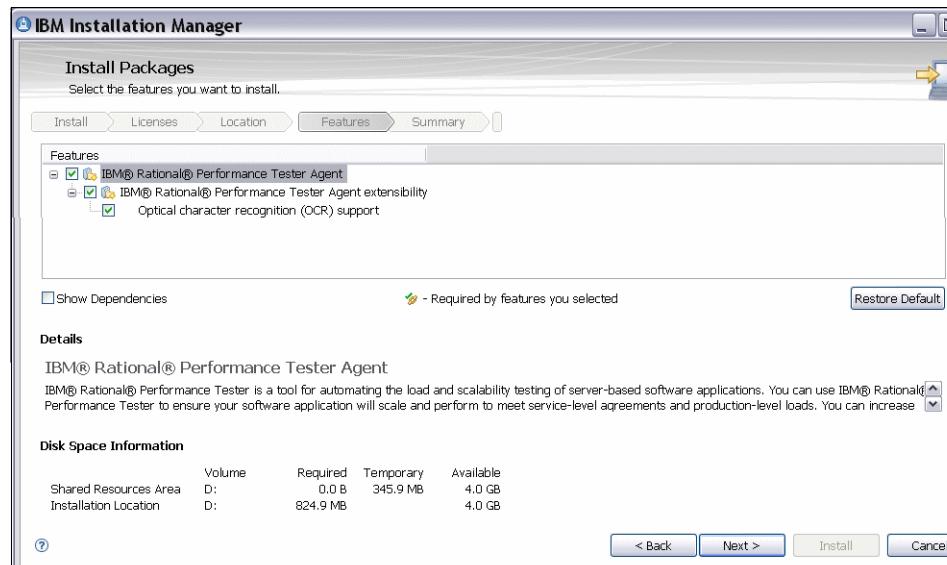


Figure A-39 Performance Tester Agent - Install Features

Step 9. Select the level of security you require for the Test Agent Controller. By default, any computer can connect and use your computer as a test agent. A good second choice is to select a custom install and list the computers that can access this test agent (Figure A-40).

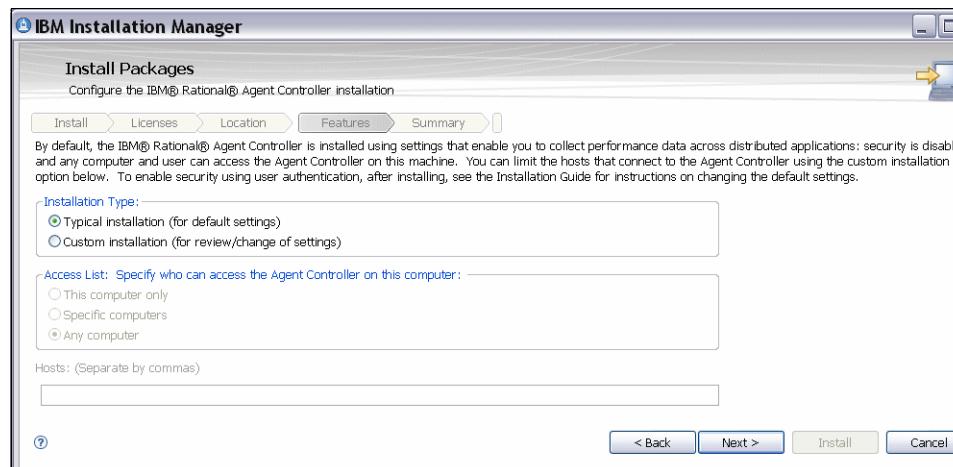


Figure A-40 Performance Tester Agent - Test Controller

Step 10. Review the features to be installed for completeness and click *Install* (Figure A-41).

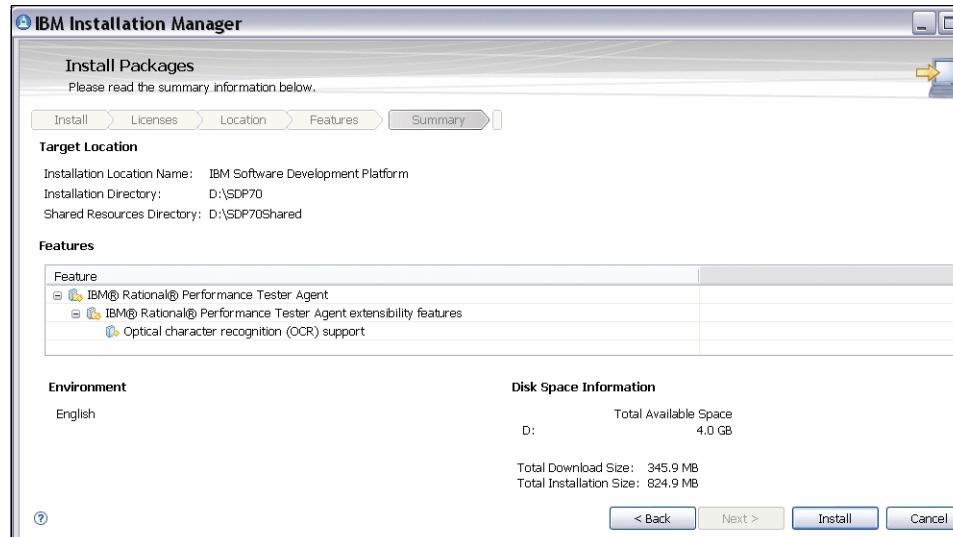


Figure A-41 Performance Tester Agent - Feature Summary

Step 11. Wait for the Installation Manager to complete the install operation (Figure A-42).

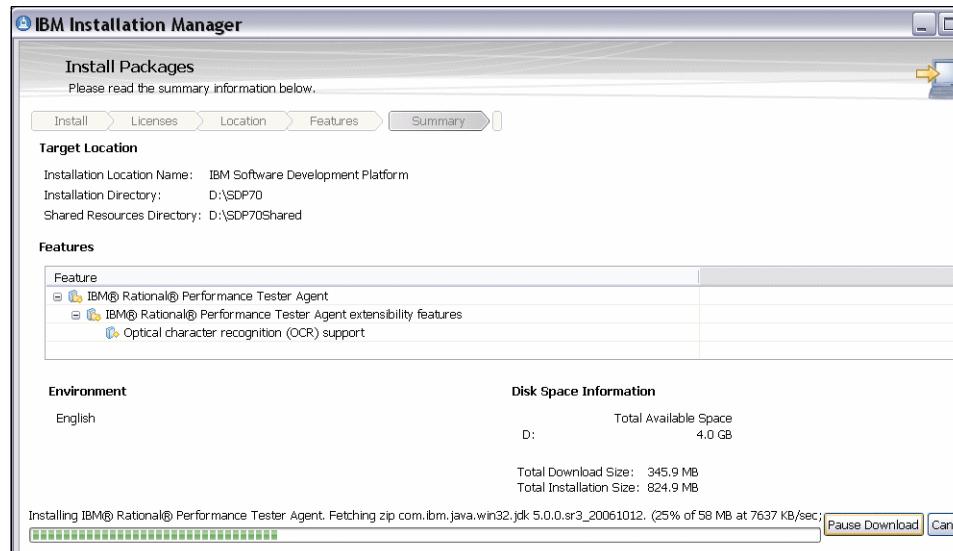


Figure A-42 Performance Tester Agent - Install Status

Step 12. Insert disk 2 when requested (Figure A-43).

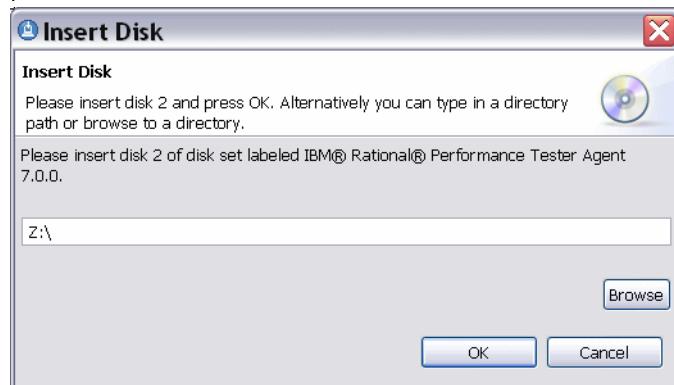


Figure A-43 Performance Tester Agent - Insert Disk dialog for disk 2

Step 13. When the install is complete, you will see a final status message indicating the success or failure of the product installation (Figure A-44).

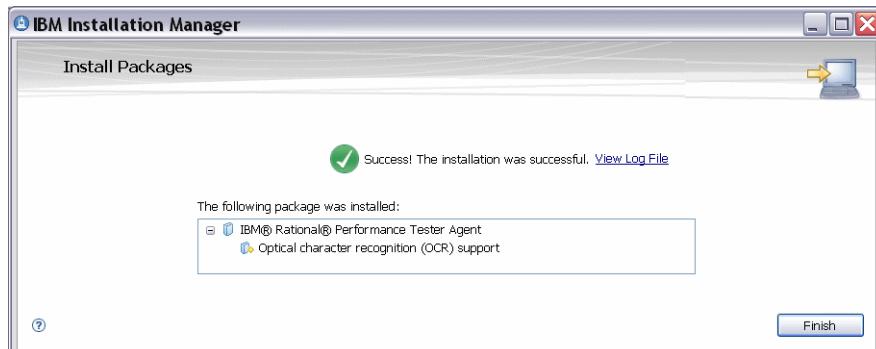


Figure A-44 Performance Tester Agent - Install Package Status

When the Test Agent has been properly installed, the ACWinService.exe process on Windows or the RAService process on Linux or UNIX should be running. This indicates that the Test Agent has been started.

To use the Transaction Breakdown Analysis feature, you should then start the data collection process (`tapmagent.exe`) running either from the Start Menu on Windows or from the Test Agent <install dir>/DCI/rpa_prod subdirectory invoke the `startDCI.sh` shell script to start the data collection (`tapmagent`) process. If this server is hosting a Web application server, you should instrument that server before starting the data collection process using the Application Server Instrumenter from the Start Menu on Windows or the `instrumentServer.sh` shell script on a Linux or UNIX system.

Product licensing

The Performance Tester product is licensed in several different ways to provide the end user with a limited term fully functional product for trial purposes as well as a permanent license capability once the product has been purchased. For the performance testing market space, this style of tool has extremely high value depending on the scale of the test in terms of virtual users emulated. The user buys only the capability that they need for their testing needs. Some vendors resell this capability on a per test environment basis. With Performance Tester, the general virtual user capability is licensed and then separate optional environments are priced and licensed separately.

Product activation kits

When the base product is installed, it comes with a 30-day trial period during which the user can experience the full functionality of the product. However, without a virtual user license pack, the user can only play back up to 5 user sessions. This mode is adequate for prospective customers to figure out if the product will meet their eventual testing needs and they can experience all of the dynamic playback and ease of use features of this new generation of performance testing tools.

When the user purchases a licensed copy of the software, it comes with a product activation kit. This can be loaded into the Performance Tester software installation through the Installation Manager's Manage Licenses function. This converts the standard 30-day trial license into a permanent license that never expires. The user's sales representative can provide an extension to the 30-day trial period through a limited term product activation kit also if the sales situation justifies a longer trial period.

Product playback licensing

When playing back a large performance test, the Performance Tester product checks out a Virtual Tester License Pack of a size larger than the number of virtual users (less the first five that are unlicensed) that you are trying to emulate. The Performance Tester console system must have the Rational License client components installed and configured to point to a Rational License server. That license server must be serving at least one Virtual Tester License Pack of a size equal to or greater than the size of the test playback attempted.

This license pack is checked out during the test playback and checked back in upon completion of the test. If no single license pack contains enough virtual testers, the Performance Tester product will attempt to combine license packs to find enough virtual testers to support the size of test playback requested. See Table A-1 for a list of all the virtual tester pack sizes.

Table A-1 Virtual Tester Pack Sizes - Floating Licenses

| | | | |
|---------------------|----------------------|------------------------|-------------------------|
| 5 virtual testers | 250 virtual testers | 2,500 virtual testers | 20,000 virtual testers |
| 50 virtual testers | 500 virtual testers | 5,000 virtual testers | 50,000 virtual testers |
| 100 virtual testers | 1000 virtual testers | 10,000 virtual testers | 100,000 virtual testers |

If you want to run more than one virtual tester of one of the product extended environments, you need a floating playback license key for that extension. The currently available product extensions are listed in Table A-2.

Table A-2 Performance Tester Product Extensions

| |
|--|
| Rational Performance Tester Extension for Siebel Test Automation |
| Rational Performance Tester Extension (for SAP systems) |
| Rational Performance Tester Extension for Citrix Systems |

Abbreviations and acronyms

| | | | |
|--------------|---|--------------|--|
| API | Application Programming Interface | IIOP | Internet Inter-Orb Protocol |
| ARM | application response measurement | IP | Internet Protocol |
| ASI | Application Server Instrumenter | IPOT | IBM Performance Optimization Toolkit |
| CA | certificate authority | ISV | Independent Solution Vendor |
| CEO | Chief Executive Officer | IT | Information Technology |
| CORBA | Common Object Request Broker Architecture | ITCAM | IBM Tivoli Composite Application Manager |
| CQEC | ClearQuest Eclipse Client | ITLM | Information Technology Lifecycle Management |
| CSV | comma separated values | ITM | IBM Tivoli Monitoring |
| DCI | data collection infrastructure | ITSO | International Technical Support Organization |
| EJB | Enterprise JavaBean | J2EE | Java2 Enterprise Edition |
| EMF | Eclipse Modeling Framework | JAR | Java archive |
| ERP | enterprise resource planning | JDK | Java Development Toolkit |
| ESP | early support programme | JITI | Just In Time Instrumentation |
| FSDM | Financial Services Data Model | JMS | Java Messaging System |
| FSS | Financial Services Sector | JNI | Java Native Interface |
| GC | garbage collection | JVM | Java Virtual Machine |
| GIF | Graphics Interchange Format | JVMPI | Java Virtual Machine Profiler Interface |
| GUI | graphical user interface | KB | kilobytes |
| GUID | global unique identifier | LAN | Local Area Network |
| HTML | Hypertext Markup Language | mB | million bytes (1,000,000) |
| HTTP | Hypertext Transfer Protocol | MB | megabytes (1,048,576) |
| HW | hardware | MCT | multi-channel transformation |
| IBM | International Business Machines | MQ | message queue |
| ICA | Independent Computing Architecture | MSDN | Microsoft Developer Network |
| IE | Internet Explorer | MSIE | Microsoft Internet Explorer |
| IFLs | Integrated Facility for Linux | MSO | modular service offering |
| IFW | Information FrameWork | ORB | Object Request Broker |
| | | OS | operating system |

| | | | |
|-------------|---|------------|----------------------------|
| PD | problem determination | XML | Extensible Markup Language |
| PDH | performance data helper | | |
| PI | profiling interface | | |
| QoS | quality of service | | |
| RAC | Remote Agent Controller | | |
| RCS | Rational Certificate Store | | |
| RFT | Rational Functional Tester | | |
| RMI | Remote Method Invocation | | |
| RPC | Remote Procedure Call | | |
| RPT | Rational Performance Tester | | |
| RSA | Rational Software Architect | | |
| RSM | Rational Software Modeler | | |
| RTBA | response time breakdown analysis | | |
| SCM | software configuration management | | |
| SDK | Software Development Kit | | |
| SDP | Software Delivery Platform | | |
| SLA | service level agreement | | |
| SOA | service-oriented architecture | | |
| SPEC | Standard Performance Evaluation Corporation | | |
| SSL | Secure Sockets Layer | | |
| STG | Systems Technology Group | | |
| SWG | Software group | | |
| TMTP | Tivoli Monitoring for Transaction Performance | | |
| TPTP | Test and Performance Tools Platform | | |
| UML | Universal Markup Language | | |
| URL | Uniform Resource Locator | | |
| VM | virtual machine | | |
| WLS | WebLogic Application Server | | |
| WMI | Windows Management Infrastructure | | |
| WSDL | Web Service Definition Language | | |

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 536. Note that some of the documents referenced here may be available in softcopy only:

- ▶ *Rational Application Developer V7 Programming Guide*, SG24-7501
- ▶ *Building Service-Oriented Banking Solutions with the IBM Banking Industry Models and Rational SDP*, REDP-4232
- ▶ *Model Driven Systems Development with Rational Products*, SG24-7368
- ▶ *Building SOA Solutions Using the Rational SDP*, SG24-7356
- ▶ *Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products*, SG24-7148
- ▶ *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297
- ▶ *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- ▶ *Best Practices for SOA Management*, REDP-4233

Other publications

These publications are also relevant as further information sources:

- ▶ IBM technote 1221972: *Rational Performance Tester 6.1.2 Workbench Memory Optimization*:

<http://www-1.ibm.com/support/docview.wss?rs=0&uid=swg21221972>

Online resources

These Web sites are also relevant as further information sources:

- ▶ IBM Rational Performance Tester:
<http://www.ibm.com/rational/>
<http://www.ibm.com/software/awdtools/tester/performance/index.html>
- ▶ Eclipse and Test & Performance Tools Platform:
<http://www.eclipse.org/>
<http://www.eclipse.org/tptp>
- ▶ SourceForge rstatd for Linux:
<http://sourceforge.net/projects/rstatd/>
- ▶ RFC standards:
<http://rfc.net/>
- ▶ Microsoft Performance Monitoring:
<http://msdn2.microsoft.com/en-gb/library/aa830465.aspx>
- ▶ Standard Performance Evaluation Corporation:
<http://www.spec.org//>
- ▶ SAP Solutions:
<http://www.sap.com/solutions/index.epx>
- ▶ Citrix:
<http://www.citrix.com>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

Acrobat Reader 411
activation kit 531
ActiveX 387
Agent Controller 33
 local 149
 problems 35
 remote 150
analysis 14
 tool 14
application
 management 256
 monitoring 256
 real-time 279
 redesign 11
 response measurement 258
 server
 instrumentation 22
 version 25
Application Server Instrumenter 264, 269
architecture 7
ARM
 agent 281
 instrumentation 284
 standard 258
authentication 21, 130
 Citrix 135
 SAP 134
automatic correlation 81
average response time 187

B

balanced system 11
bandwidth 24
Banking Data Warehouse 473
BEA WebLogic Application Server 261
best practices
 SAP 400
bottleneck 11
branch transformation 474
browser
 caching 37
 configuration 30

 settings 30
business
 analyst 6
 conditions 14
 continuity 475
 intelligence 475
 transaction 17, 256
 workflows 19
byte-code instrumentation 261

C

caching 20
capacity 16
 measurement 23
capacity testing 14
changing hosts 128
Citrix 38
 authentication 135
 change host 130
 condition 114
 delay 108
 desktop session 408
 guidelines 404
 loop 111
 performance report 420
 performance test 403
 playback 425
 Recorder 406
 Control Window 410
 recording 405
 session 417
 test 62
 complex 424
 edit 414
 optimizing 425
 script 406
 timing 123
 verification point 101
 verification point report 420, 422
 Windows environment 402
Citrix Presentation Server 402
clock
 synchronize 216

command line interface 153
comment element 135
Common Object Request Broker Architecture 259
common problems
 SAP 400
competition 5
condition 111
 Citrix 114
 HTTP 114
 SAP 114
confidence interval 184
connection 120
 pooling 22
 refused 150
 SAP 59, 127
 think time 107
console 322
controller 322
cookie cache 38
correlated data 418
correlation
 automatic 81
 manual 83
 multiple results 192
 preferences 82
 problems 86
 time 183
counter
 add 196
 generic 192
CPU
 usage 333
 usage measurement 347
 utilization 23
critical business systems 4
CSV file format 64
custom code 7, 430
 add 117
 add element 432
 data area 443
 datapool 452
 duration control 461
 element 432
 example 435
 HTTP response 457
 import Java code 448
 Java code 434
 loop 445
 passing data 457
 return data 459
 reuse 119, 450
 test architecture 431
 transaction 445
 usage 430
custom transaction 115
custom verification 104
customized reporting 7

D

data
 area 443
 collection 10, 22, 217
 infrastructure 269, 279, 282
 collision 21
 correlation 9, 42, 81
 Siebel 56
 strategy 85
database
 connection 22
datapool
 add to test 70
 candidate 418
 create 66
 custom code 452
 digital certificates 80
 editor 68
 import 67
 login names 395
 options 72
 overview 65
 Siebel 56
 substitution 78
 variable 70
DB2
 instrumentation 284
deadlock 294
debugging 20, 86, 148
 launch 148
delay 107, 120
 Citrix 108
 SAP 123
 schedule 163
description element 136
digital certificate 80
 HTTP 131
 store 80
disabling security warnings 33

distributed
application
analysis 14
business transactions 256
enterprise systems 9
environment 9
playback 10
distribution
Student t 183
driver 322
architecture 325
configuration 338
CPU usage 358
engine overhead 332
local 353, 365
maximum JVM heap size 364
measurement 337
memory 355
remote 364
sizing 355
virtual user capacity 356, 359
dynamic
discovery 282
load generation 14
workload implementation 471

E

Eclipse 8
Modeling Framework 213, 322
Test & Performance Tools Platform 8
Test and Performance Tools Platform 64
user interface 8
encryption 35
end-to-end transaction 257
engine architecture 325
enterprise system 5
environment variable 32
equipment 14
execution rate 167
Execution Statistics views 303
expected behavior 90
extensibility 7

F

Fair Isaac TRIAD 473
firewall 25
exception 151

G

garbage collection 24, 327
generic counter 192
global
economy 5
unique identifiers 260
goals 6
graceful stop 147
graphical analysis 300

H

hard stop 147
header 124
headless test 463
heap
allocation 327
fragmentation 22, 25
limit 326

HTTP

authentication 131
change host 128
condition 114
connection 123
delay 122
header 124
memory footprint 366
percentile report 24, 201
recording 37
request rate 361
response
custom code 457
element 48
test 53
loop 109
test generation 39

I

ICA Client 402
image
synchronization 420
incremental virtual user CPU utilization 322, 335, 348
incremental virtual user memory footprint 322, 329–330
Independent Computing Architecture 62, 402
Information Technology Lifecycle Management 279
infrastructure simplification 476

injector 322
input data variation 20–21
installation 504
Installation Manager 505
instrumentation 22, 261
instrumenting
 GUI 271
interactive graphical analysis 300
Internet
 options 30
Internet Explorer 30
 proxy 31
 security warnings 33
InternetShoppingCart workload 344
InternetShoppingHome workload 345
IP alias 173
IPOT agent 280

J

J2EE monitoring component 261
Java
 applet 7
 memory management 322
 considerations 326
 programming language 8
Java Native Interface 218
Java Virtual Machine Profiling Interface 262
JITI 261
JRockit VM 270
Just In Time Instrumentation 261
JVMPi 262
 agent 281

K

KeyTool program 132

L

legacy system 6
Linux
 environment variable 32
 performance counters 223
 rstat daemon 225
 rstatd 222
load
 balancer 21
 balancing 25
 generation 14

range testing 16
location 160
lock-step 332
log
 data collection 171
 level 171
 message 148
 problem determination 176, 438
 test debugging 437
 test verdict reporting 437
loop 108
 Citrix 111
 custom code 445
 HTTP 109
 SAP 110
 schedule 161

M

manual correlation 83
measurement 18
 infrastructure 216
memory
 allocation 23
 error 152
 footprint 322
 HTTP 366
 measurement 339
 leak 22, 25
 measurement 328
 optimization 321
Method Invocation Details view 305
metrics 18
Microsoft Developer Network 253
Microsoft Remote Desktop Protocol 402
monitoring
 agent 234
Mozilla
 environment variables 30, 32
multi-channel transformation 474
multiple hosts 173
multi-user playback 39
 SAP 394

N

NetWeaver 384
network
 topology 25

O

Object Request Broker 259
OMEGAMON 253
operating system monitoring 14
optimize key strokes 426

P

paced loop 330
page
 element 23
 title 90
paging 24
peak load 17
PeopleSoft 64
percentile report 24, 201
performance
 analysis
 document 27
 counters 216
 data
 helper 217
 measurement 182
 problems 294
 report 24
 requirement 18
 results 182
 schedule 155
 configure 213
 test plan 27
 tester 18
 testing 4
 capability 13–14
 methodology 26
 process 11, 13, 15
 tool 5
 time range 199
Plants41k workload 341, 367
platform independence 7
playback 9, 140
 architecture 10
 command line 153
 engine 7
 Java process 152
 licensing 531
 multi-user 39
 problems 148
 SAP 396
 schedule 142

status 143
stop 147
pmirm.xml 267
policies 312
pool
 connection 24
portmapper daemon 226
Presentation Server
 Citrix 402
private bytes 328
probe 261
problem determination
 level 175
 log 148, 176, 438
process
 architecture 323
 performance testing 15
product
 activation kit 531
 licensing 530
production
 cutover 5
productivity 6
profit target 5
protocol extension license 39
proxy
 server 31, 36
 settings 32

Q

queries 234

R

ramp-up period 22
random arrival 182
random selector 164
RAService.exe 325
Rational Certificate Store 132
Rational ClearCase 52
Rational ClearCase LT 52
Rational ClearCase SCM Adapter 52
Rational ClearQuest Eclipse Client 464
Rational Functional Tester 395
Rational License Server
 installation 518
Rational Performance Tester
 architecture 7
 goals 6

installation 509
KeyTool 134
launchpad 504
overview 4
project 46
Rational Performance Tester Agent 150
 installation 523
Rational Performance Tester Extensibility Software Development Kit 53
Rational Performance Tester Extension for Citrix Presentation Server 401
readability 137
real-time
 application monitoring 279
 observation 241, 288
recorder
 certificate 33
 version 36
recording 9, 29
 file 36
 HTTP 37
 tips 37
Redbooks Web site 536
 Contact us xviii
reference 83
regulatory compliance 475
reliability 16
Remote Service Monitor 224
report 24
 analysis 295
 customizing 248
reporting 182
requirement 18, 21
resource
 data 177
 monitoring 10, 212
 data sources 177
 platform 212
 utilization 11
response
 code 91
 content 93
 delay 122
 size 92
 time 10, 98
 breakdown 178, 285
 breakdown analysis 205
 measurement 14
RPT Tuning Tool 484
RSM 224
rstatd 206, 222
run
 analysis 11
 location 160

S

sampling rate 173
SAP
 authentication 134
 best practices 400
 change host 129
 common problems 400
 condition 114
 connection 59, 127
 datapool 58
 delay 123
 enterprise application 383
 event 60
 events 58
 GUI for Java 385
 GUI for Windows 383, 385
 GUI library 399
 International Demonstration and Education System 385
 login datapool 395
 loop 110
 multi-user playback 394
 NetWeaver 384
 performance test 389
 playback 396
 Protocol Data view 59, 99
 screen details 60
 test 58
 complex 397
 editor 393
 execute 396
 recording 389
 think time 121
 verification point 96
scalability 325
scalable 7
scale factor 198
schedule 21
 add delay 163
 add loop 161
 add random selector 164
 add test 161

behavior 165
create 140, 156
cut and paste 165
duration 170
elements 157
execution rate 167
IP alias 173
measurement 339
playback 140
resource monitoring 177
statistics interval 144
statistics log level 179
stop 170
test 155
think time 166
user behavior 167
screen capture 419
search
 action 50
 history 50
 match 50
 results 51
Search view 49
Secure Sockets Layer 53
secure Web site 33
security
 warnings 33
server.xml 266
service level agreement 256
serviceconfig.xml 35
servicelog.log 35
session ID 81
Siebel 38
 content verification 96
 Data Correlation engine 56
 Data Correlation Library 57
 data sources 56
 star arrays 56
 test 55
Siebel Business Analytics 473
sizing guidelines 321, 351
Smart Bank 471
 implementation 484
 showcase 472
socks protocol 34
Software Delivery Platform 327
SourceForge 225
Sovereign JVM 326
stability 23
standard deviation 188
Standard Performance Evaluation Corporation 361
state
 management 464
 transition diagram 465
statistics interval 144
statistics logging 179
steady state 23, 182, 184
Stonesoft Stonegate 473
stop test run 147
strategic advantage 4
stress testing 16
Student t distribution 183
substitution 74
swapping 24
SWEAT 14
synthetic workload
 simulation 463
 testing 431
system
 architect 18
 architecture 7
 capacity 16
 design 18
 down time 23
 functionality 42
 performance 11
 tuning 14–15

T

table lock 21
technology
 challenge 6
test
 add datapool 70
 agent 10, 322
 Citrix 62, 403
 conditional logic 111
 custom code 79
 data 42
 datapool 56
 debugging 437
 editing 42
 best practices 46
 editor
 preferences 44
 element
 add 105

execution result 22
final report 25
generation 29, 38
 HTTP 39
goals 16
headless 463
HTTP 53
log 442
 level 145
plan 27
playback 9, 39, 140
readability 42, 137
recorder 20
result 27
reuse 135
run monitoring 10
SAP 58
SAP GUI for Windows 385
scenario 27
schedule 21, 42, 155
script 19
search 46
Siebel 55
substitution 74
success 19
tool asset 27
verdict reporting 437
verification 89
version 52
Test & Performance Tool Platform 322
Test & Performance Tools Platform 213
 see TPTP
testing
 artifact 26
 capability 14
 goal 25
 interval 11
 iterative 15
 laboratory 5
 problem space 4
 process 13, 15
 tool platform 8
 Web-based 6
think time 120, 166
throughput 186
time
 correlation 183
 range 199
timed stop 147
timing 42, 120
 Citrix 123
Tivoli ARM engine 259, 280
Tivoli Composite Application Manager 178, 256
Tivoli Composite Application Manager for Response
Time Tracking 259, 306
Tivoli Composite Application Manager for
WebSphere 306
Tivoli Data Warehouse 230
Tivoli Enterprise Console 231
Tivoli Enterprise Monitoring Server 230
Tivoli Enterprise Portal Client 230
Tivoli Enterprise Portal Server 230
Tivoli J2EE Monitoring Component 261, 264
Tivoli Monitoring 177, 206
 architecture 229
 import data 243
 performance schedule 238
 queries 234
Tivoli Monitoring for Transaction Performance 259
tool
 architecture 9
 chain 9
 framework 8
 platform 8
tools 14
TPTP 8, 64
transaction
 correlation 259
 custom 115
 custom code 445
 element 114
 flow 280
 monitoring 256
 rate 22, 203
 report 24
 throughput 10, 186
 timing 208
traps 312
troubleshooting 35
 data correlation 86
true response time 185
tuning 15, 24
 recommendation 27

U

UML

Interaction view 302

Sequence Diagram view 303
undo 44
uninstrument
 command line 277
 GUI 278
UNIX
 performance counters 223
 rstatd 222
usage pattern 9
use case 19
user
 behavior 167
 group 157
 add 159
 remote location 168
 scenario 26
 session 20
 think time 26

V

variables.xml 265
vendor independence 6
verification 89
 point 11, 90
 Citrix 101
 page title 90
 report 24
 response code 91
 response content 93
 response size 92
 response time 98
 SAP 96
 SAP screen title 97
 Siebel content 96
 Window title 102
virtual user 22
 add 146
 CPU utilization 322
 IP alias 175
 memory footprint 322
virus detection 151

W

Web-based testing 6
WebLogic
 instrumentation parameters 273
WebLogic Application Server 261, 267
WebSphere Application Server 264

instrumentation parameters 272
Window
 events 417
 verification points 423
Windows Management Infrastructure 216
Windows Performance Monitor 206, 216
 performance schedule 220
workbench 323
 architecture 324
 console 353
 memory 354
 memory optimization 321
 sizing 354
worker threads 325
working set 328
workload 14–15
 analysis 43, 479
 definition 22
 injection 497
 measurement 15, 338
 model 17, 21
 specification 26

IBM



Redbooks

Using Rational Performance Tester Version 7

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages

**Redbooks**

Using Rational Performance Tester Version 7

Introducing Rational Performance Tester

This IBM Redbooks publication is intended to show customers how Rational processes and products support and enable effective systems testing.

Understanding RPT functions and features

The book describes how performance testing fits into the overall process of building enterprise information systems. We discuss the value of performance testing and its benefits in ensuring the availability, robustness, and responsiveness of your information systems that fill critical roles for your enterprise.

Applying RPT to enterprise application testing

Based on years of project experience, we describe the key requirements needed in performance testing tools and how the IBM Rational Performance Tester tool was developed to meet those requirements. We also walk through the choices that we made to steer the tool architecture into using an open source platform as its base with Java as its language to permit ubiquitous platform support.

This book is structured into two parts:

- ▶ Understanding Rational Performance Tester
- ▶ Applying Rational Performance Tester to enterprise application testing

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**